POLITECNICO DI TORINO

*Department of Mechanical and Aerospace Engineering*

Master of Science in Automotive Engineering


# Path planning for an autonomous racecar with Rapidly-exploring Random Tree

Supervisor: Prof. Nicola AMATI

Co-supervisor: Prof. Andrea TONOLI

Prof. Angelo BONFITTO

Eng. Stefano FERACO

Eng. Sara LUCIANI

Candidate:

EUGENIO TRAMACERE

257055

Academic Year 2020 - 2021

**Abstract**

Among the considerable technical problems that self-driving vehicles have to face, there is the path planning, that is a function for autonomous vehicles. Path planning is defined as the problem of finding a continuous collision free path from an initial configuration to a predetermined goal. In this context, the present thesis work focuses on the design and implementation of a real-time local trajectory planning method using Rapidly-Exploring Random Tree Algorithm based on Dubins curves for an AWD electric racing vehicle in different scenarios, straight, left and right bends. The key contribution of this work is to present a framework able to generate a feasible free collision path considering differential constraints for non-holonomic car-like robot and its real-time computation ability in solving the dynamic motion planning problem. As a first step, a preliminary research on motion planning techniques is carried out. Subsequently, the problem is stated, analysing the structured environment in which the race car has freedom of action. Afterwards, the methodology is defined and the final configuration, i.e. the goal, is extrapolated from the local map coming from the perception pipeline, exploiting the functionality of a LiDAR sensor mounted onto the front wing of the racing vehicle. The vehicle is considered as a three Degree-of-Freedom bicycle dynamic model and a Stanley Controller is implemented to control the lateral and longitudinal vehicle dynamics. The feasibility of the planned trajectory is also evaluated with respect to command signals for the steering and acceleration actuators featured by the retained racing vehicle. Conclusions with critical comments about the obtained results and the possible future works perspectives complete the present thesis work.

# Contents

Chapter 1

# Introduction

In recent years the concept of autonomous driving has become a recurring debate. *Self-driving* vehicles are now being talked about on a daily basis. Academic and industrial environments are in turmoil and numerous research activities are taking place during these times.
This work, carried out in the LIM laboratory inside the Polytechnic of Turin, is part of the project to convert the *SC19* prototype of the *SquadraCorse Polito* student team into an autonomous vehicle in accordance with the Formula Student regulations, capable of competing in the Driverless category.

## 1.1 Background

Every year a considerable number of people are involved in a road accident, sometimes in a serious way, others unfortunately facing death. The number of people killed in road crashes around the world continues to be high, even if objectively modern vehicles are safer than those produced in the past decades both for people inside the vehicle and for external road users such as pedestrians or cyclists.
According to the World Health Organisation's "Global Status Report on Road Safety" [36], every year 1.2 million died in road accidents and between 20 and 50 million suffer non-fatal injuries. Over 90% of the world's fatalities on the roads occur in low-income

and middle-income countries, which have only 48% of the world's registered vehicles. On the other hand, the death rates due to road accidents have decreased in the last 4-5 decades in high-income countries even if road traffic injuries remain an important cause of death, injury or disability.

Compared to the global trend, in Europe , as can be seen from Figure 1.1, the number of deaths due to road accidents fell by 43% in the period from 2001 to 2010 and by another 21% in the period from 2010 to 2018 [21]. However, in 2018 alone, 25,100

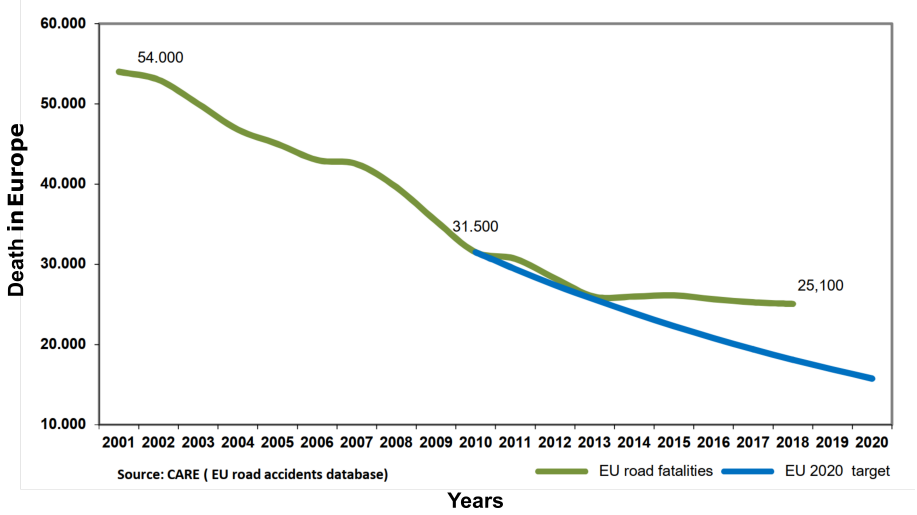



**Figure 1.1:** *Europe accidents trend in the last decades.*

people lost their lives on EU roads and 135,000 were severely injured. All this has an unacceptable social impact to pay for mobility. In merely economic terms for the European society, 280 billion has been estimated as annual cost due to road accidents, equivalent to about 2 % of the EU GDP (Gross Domestic Product) [9].

In 2019, 172,183 road accidents occurred in Italy resulting in death or injury, slightly down comparing with 2018 (-0.2%), with 3,173 deaths and 241,384 injured (-0.6%). Among the victims, the number of cyclists (253;+15.5%) and motorcyclists (698;+1.6%) increased, while pedestrians (534;-12.7%), moped users (88;-18.5%), trucks occupants (137,-27.5%) and passenger cars users (1,411;-0.8%) showed a decrease. The increase of victims among cyclists, mainly on primary roads, in builtup and outside urban area, is also associated to a growth in road accidents involving bicycles (+ 3.3%), to the spread of the two-wheels vehicles use for daily journeys, equal to 25% in 2019 and to the increase in bicycles sales in 2019, 7% more than in 2018. The social cost of road

accidents in 2019, calculated on the basis of parameters indicated by the Ministry of Infrastructure and Transport is equal to 16.9 billion euros, 1% of the national GDP [20].

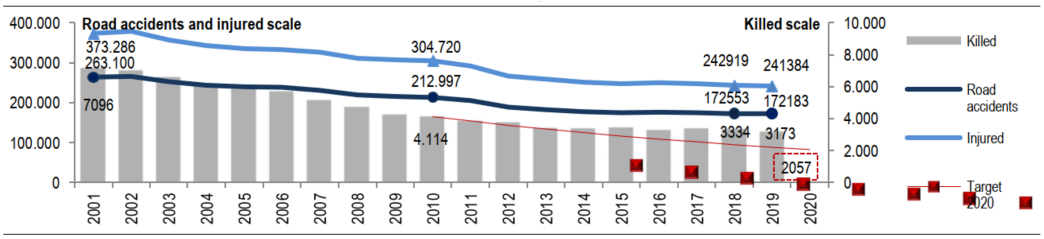



**Figure 1.2:** *Road accidents in Italy resulting in death or injury, killed and injured from 2001 to 2019.*

In most cases, these accidents are due to driver's fault, therefore s/he could be theoretically replaceable by self-propelled cars. Research experiments on autonomous vehicles became highly progressive through the continuous effort of department engineers and graduates working for various competitions and company projects.

## 1.2 SAE Driving Automation Levels

The SAE Recommended Practice (J3016_201806) describes motor vehicle driving automation systems that perform part or all the *dynamic driving task* (DDT) on a sustained basis [19]. The Society of Automotive Engineers provides a taxonomy with detailed definitions of six levels of driving automation, ranging from no driving automation (level 0) to full driving automation (level 5).

As reported in Figure 1.3, the level of automation is defined by reference to the specific ***role*** played by each of the three primary actors in performance of the DDT and/or DDT fallback. The three primary actors in driving are the *(human) user*, the *driving automation system*, and other *vehicle systems and components*.

The term ***role*** in the Recommended Practice [19], refers to the expected role of a given primary actor, based solely on the design of the *driving automation system*. For example, a driver who fails to monitor the roadway during engagement of a level 1 adaptive cruise control (ACC) system still has the role of driver, even while s/he is neglecting

| Level | Name | Narrative definition | DDT | | DDT fallback | ODD |
|---|---|---|---|---|---|---|
| | | | Sustained lateral and longitudinal vehicle motion control | OEDR | | |
| ***Driver* performs part or all of the *DDT*** | | | | | | |
| 0 | **No Driving Automation** | The performance by the *driver* of the entire *DDT*, even when enhanced by *active safety systems*. | *Driver* | *Driver* | *Driver* | n/a |
| 1 | **Driver Assistance** | The *sustained* and *ODD*-specific execution by a *driving automation system* of either the *lateral* or the *longitudinal vehicle motion control* subtask of the DDT (but not both simultaneously) with the expectation that the *driver* performs the remainder of the *DDT*. | *Driver* and *System* | *Driver* | *Driver* | Limited |
| 2 | **Partial Driving Automation** | The *sustained* and *ODD*-specific execution by a *driving automation system* of both the *lateral* and *longitudinal vehicle motion control* subtasks of the *DDT* with the expectation that the *driver* completes the *OEDR* subtask and *supervises* the *driving automation system*. | ***System*** | *Driver* | *Driver* | Limited |
| ***ADS* ("*System*") performs the entire *DDT* (while engaged)** | | | *System* | ***System*** | *Fallback-ready user (becomes the driver during fallback)* | Limited |
| 3 | **Conditional Driving Automation** | The *sustained* and *ODD*-specific performance by an *ADS* of the entire DDT with the expectation that the *DDT fallback-ready user* is *receptive* to *ADS*-issued *requests to intervene*, as well as to *DDT performance-relevant system failures* in other *vehicle* systems, and will respond appropriately. | | | | |
| 4 | **High Driving Automation** | The *sustained* and *ODD*-specific performance by an *ADS* of the entire *DDT* and *DDT fallback* without any expectation that a *user* will respond to a *request to intervene*. | *System* | *System* | ***System*** | Limited |
| 5 | **Full Driving Automation** | The *sustained* and unconditional (i.e., not *ODD*-specific) performance by an *ADS* of the entire *DDT* and *DDT fallback* without any expectation that a *user* will respond to a *request to intervene*. | *System* | *System* | *System* | **Unlimited** |

**Figure 1.3:** *Levels of Driving Automation according to SAE J3016_201806.*

it. The Dynamic Driving Task (DDT) is referred to all the real-time operational and tactical functions required to *operate a vehicle* in on-road traffic such as:

1. *Lateral vehicle motion control* via steering;
2. *Longitudinal vehicle motion control* via acceleration and deceleration;
3. *Monitoring* the driving environment via object and event detection, recognition, classification and response preparation;
4. Object and event response execution;
5. Maneuver planning;
6. Enhancing conspicuity via lighting, signaling and gesturing, etc.

Subtasks (3) and (4) are referred to collectively as *object and event detection ad response* (OEDR) while the DDT Fallback represents the response by the user to either perfom the DDT or achieve a *minimal risk condition* after occurence of a DDT

performance-relevant system failure or upon *operational design domain* (ODD) exit. The Operational Design Domain (ODD) are the operating conditions under which a given *driving automation system* or *feature* thereof is specifically designed to function [19]. Therefore the various levels of driving automation are:

1. Level 0: No driving automation. Both longitudinal and lateral maneuvers are completely performed by driver even though the vehicle is equipped with active safety systems, such as ABS or ESP;

2. Level 1: Driver assistance. The vehicle's System is capable enough to perform either one of the maneuvers (lateral or longitudinal), but not both simultaneously. The driver will have manual monitoring control during the system action and perform other tasks (eg. ACC);

3. Level 2: Partial Driving Automation. The vehicle's *Autonomous Driving* (AD) system is capable enough to perform both lateral and longitudinal maneuvers under the complete supervision of the driver. The driver takes action in cases of any vulnerability of the AD system in doing the driving tasks;

4. Level 3: Conditional Automation. The vehicle's AD system is capable enough to take full control over certain scenarios and in case of any malfunction fall back option warns the driver to take full control manually. The driver's full attention is required during the action;

5. Level 4: High Driving Automation. The vehicle's AD system is capable enough to perform the full driving task over limited scenarios and it is capable enough to handle high constraints and there won't be any need in driver's intervention;

6. Level 5: Full Driving Automation. The vehicle's AD system will have capabilities to function in all scenarios and there won't be any contingency safety system needed in case of critical situations.

## 1.3 AVs Benefits and Disadvantages

In this context of greater awareness of the social impact that road accidents have every year, autonomous driving techonologies seem to be the best solution together with the modernization of the road infrastructures. Technologies for self-driving vehicles have received considerable interest in the academic, industrial and military domains especially in the last 20 years, as an example the DARPA Grand Challenge [5] and the Urban Challenge [6] that pushed the robotics community to build autonomous cars for unstructured or urban scenarios.

Car manufacturers have also followed this trend, mainly driven by the possibility of increasing their revenues as AVs open new horizons in the world road mobility. They started and are still investing on the development and implementation of Advanced Driver Assistance System (ADAS) and the development of technologies suitable for autonomous driving vehicles (AV).

Many decision-makers and practitioners wonder how *self-driving* vehicles (AVs) will affect future travel, and therefore the need for roads, parking facilities and public transit services. Optimists predict that by 2030, autonomous vehicles will be sufficiently reliable, affordable and common to displace most human driving, providing huge savings and benefits. There is considerable uncertainty concerning autonomous vehicle development, benefits and costs, travel impacts, and consumer demand. Considerable progress is needed before autonomous vehicles can operate reliably in mixed urban traffic, heavy rain and snow, unpaved and unmapped roads, and where wireless access is unreliable [31]. Years of testing and regulatory approval will be required before they are commercially available in most jurisdictions. The first commercially available autonomous vehicles are likely to be expensive and limited in performance. They will introduce new costs and risks and these constraints will limit sales. Many motorists will be reluctant to pay thousands of extra euros for vehicles that will sometimes be unable to reach a destination due to inclement weather or unmapped roads.

Figure 1.4 illustrates autonomous vehicle user costs comparing *Human Driven* and *Autonomous* vehicles. They are likely to cost more than human-driven private vehicles and public transit, but less than human-driven taxis and ridehailing services.This

is probably one of the reasons that prompted Google to develope and test a fleet of cars since 2009 with *Google Self-Driving Car* project and create the *Waymo* society, specialized on AV technologies [31]. Shared autonomous vehicles will be cheaper but less convenient and comfortable than private AVs, so many households, particularly in suburban and rural areas, will own their own.

From *User impacts* point of view, AVs potential benefits could be the reduction of

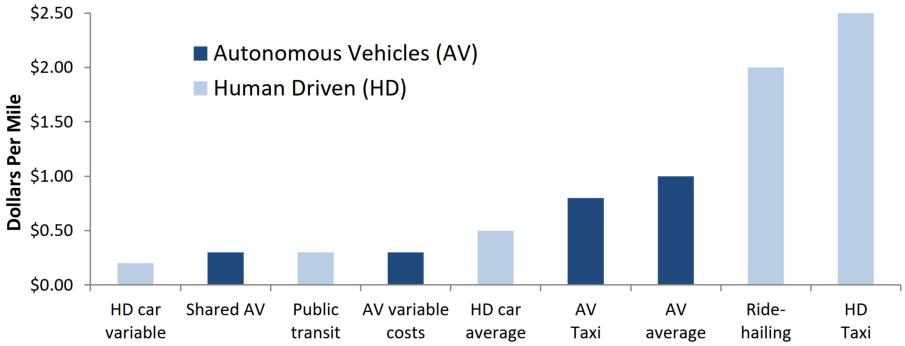



**Figure 1.4:** *Costs per Mile for Autonomous and Human Driven vehicle.*

driver's stress and the increase of productivity (play and work while travelling) or the reduction of paid driver cost (cost decrement for taxis services and commercial transport drivers). On the contrary the AVs require additional equipment, services and fees and therefore they cost more than HD vehicles. Users might be also exposed to additional crashes caused by system failure, platooning, higher traffic speeds or experience reduced security and privacy; AVs may be vulnerable to information abuse (hacking), and features such as location tracking and data sharing may reduce privacy.

Seeing the *Impacts on others*, AVs may reduce crash risks and insurance costs. Furthemore self-driving vehicles may increase the road capacity and cost savings since more efficient vehicle traffic may reduce congestion and roadway costs. Moreover, energy consumption and pollution are positively affected with a consequent increase of fuel efficiency and reduction of emissions.

## 1.4 Thesis Motivation

The motivation of this project is to convert an AWD single-seater racecar into a self-driving vehicle competing in the world's largest racing competition, Formula SAE, *Driverless* category. The starting point is the *SC19* protype, winner of the 2019 Formula ATA in class 1E, reserved for electric cars. In the process of building an autonomous system, there are three major division and they are *Perception*, *Motion Estimation and Mapping*, and *Control*. This thesis mainly focuses on the second and third part. The objective of the analysis is to design an effective *Local Path Planner* able to guide the vehicle in each *Mission* it has to perform during the competition.

## 1.5 Thesis Outline

The Thesis is organized as follows:

1. *Chapter 2*: It presents the Formula Student Driverless Competition and then the Thesis Problem Statement, the *Acceleration Mission*. Secondly, the Hardware concept of the SC19 racecar is discussed.

2. *Chapter 3*: Firstly the method by which the frames coming from the Lidar sensor can be used to automatically define the *goal* is explained. The methodology that leads to the construction of the Dubins curve-based RRT algorithm and the extrapolation of the best trajectory for each frame are defined. Then it is presented the Simulink™ model and a detailed description of the controller and vehicle model used for the Problem Statement.

3. *Chapter 4*: It presents the results coming from the simulation of the Simulink™ model. A detailed analysis of them is also present.

4. *Conclusions*: It presents the conclusions and future works of this project.

Chapter 2

# Formula Student Driverless Competition

Formula SAE is an international university engineering design competition initially proposed by the Society of Automotive Engineers (SAE) which involves the design and production of a racing car, evaluated during a series of tests based on its design qualities and engineering efficiency. Until 2016, the categories of each competition were 2: Class 1C (combustion engine vehicles), Class 1E (for electric vehicles). One year later, Formula student Germany has added a third category, Class 1D, for self-driving vehicles. In fact, recognizing the interest in autonomous driving and the associated technological challenges, Formula Student Germany (FSG) organized the first driverless competition, followed by other countries in 2018 (Italy, UK and Hungary). The new, future-oriented competition fronts students with a completely new challenge. They are to develop a race car that can run without a driver in *Autonomous* mode, or with a driver in a *Manual* mode [14]. The vehicles so must meet the technical requirements of the Class to which the type of vehicle tractive system refers and at the same time the technical requirements of the self-driving vehicles Class.

Which driverless car will win the FSD competition will not only be decided on basis of pure autonomy and correlated performance. As in the existing competition classes, the combines static and dynamic events are what counts for the victory. To minimise any risks, the autonomous race cars compete in a secured, person-free test area. The

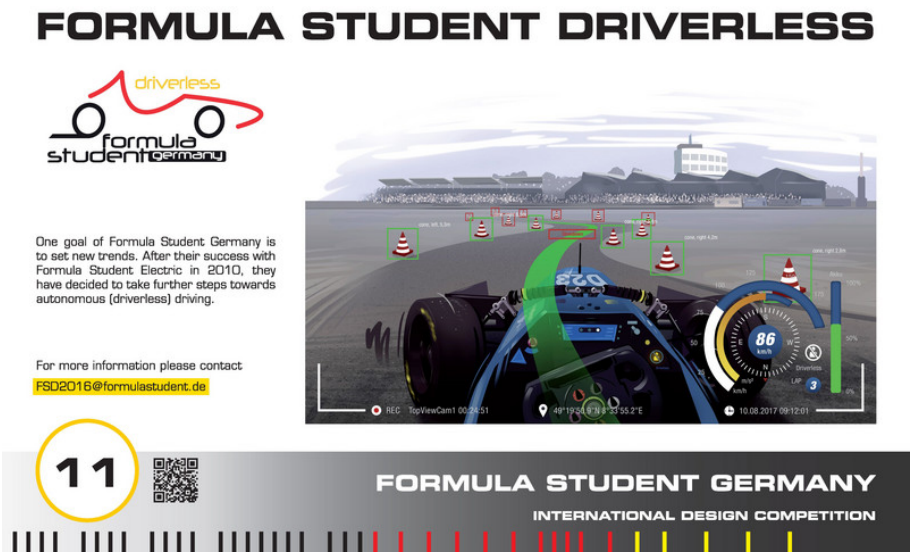



**Figure 2.1:** *Formula Student Driverless (FSD) competition class.*

most challenging part is that the layout of the racetrack is not known a priori [23]. In fact, in the the first lap, the vehicle has to be able to move within the boundary of the track thanks to the informations that come from the stereo-camera and the LiDAR. On the base of these data, the trajectory planning strategy is to calculate the path the vehicle has to follow, of course complying with its dynamic behavior in order to avoid vehicle dynamic instability or failure of the system. After the vehicle arrives at the finish line by covering the first lap, or in other words, it passes over a point it had previously travelled, the circuit map is saved in the *AV system* memory. In fact, in the laps following the first one, the vehicle already knowing the track and the trajectory to follow, adopts an aggressive driving strategy aimed at minimizing lap time. The one proposed above is the main race, *Trackdrive*. It consists of completing ten laps, as fast as possible, around an unknown track defined by small 228 $\times$ 335 mm cones. Blue and yellow cones are used to distinguish the left and the right boundary respectively. In the following chapters the various *Missions* to which the vehicle is subjected will be treated individually.

**Table 2.1:** *FSD Acceleration Event podium in FSG 2019 competition.*

| **FSD19 Scoring Results Acceleration** | | | | |
|---|---|---|---|---|
| Car | City / University | BestTime [s] | Scores (max 75) | Placing |
| 433 | Zürich ETH | 3,597 | 75 | 1 |
| 466 | Augsburg UAS | 4,056 | 58,82 | 2 |
| 519 | Roma U Sapienza | 5,224 | 30,46 | 3 |

## 2.1 Problem Statement

The *Trackdrive* mission was defined in the previous paragraph, at page 10. The thesis work, instead, was focused initially on the *Acceleration* mission. According to the "Formula Student Rulebook 2020" [39], the D 5.3 prescribes the Acceleration Procedure for Driverless Vehicles. It is composed of three phases:

1. *Staging*: The foremost part of the vehicle is staged at 0.30 m behind the starting line. Vehicles will accelerate from a standing start.

2. *Starting:* A go-signal from RES is used to indicate the approval to begin, timing starts only after the vehicle crosses the starting line and stops after it crosses the finish line.

3. *Stopping*: After the finish line, the vehicle must come to a full stop within 100 m inside the marked exit lane and enter the finish-state.

The acceleration track, according to the D5.1 in "Formula Student RuleBook 2020" [39], is a straight line with a lenght of 75 m from starting line to finish line. The track is at least 5 m wide. Cones are placed along the track at intervals of about 5 m. Cone locations are not marked on the pavement. The minimum track width is 3 m, only for Driverless Vehicle *Acceleration Event*. In Figure 2.2 it is reported the layout track for the *Acceleration Procedure*.

Looking at the results of the *Acceleration* Event for the Formula Student Driverless category (FSG competition) of 2019, in order to aim for victory, the SC19 racecar has to reach an average speed during this event of at least 20 $ms^{-1}$, Table 2.1.

Regarding the scores, according to the D5.4 contained in FS Rulebook [39], 3.5 points are given to each team that finishes at least one run without a DNF or a disqualification

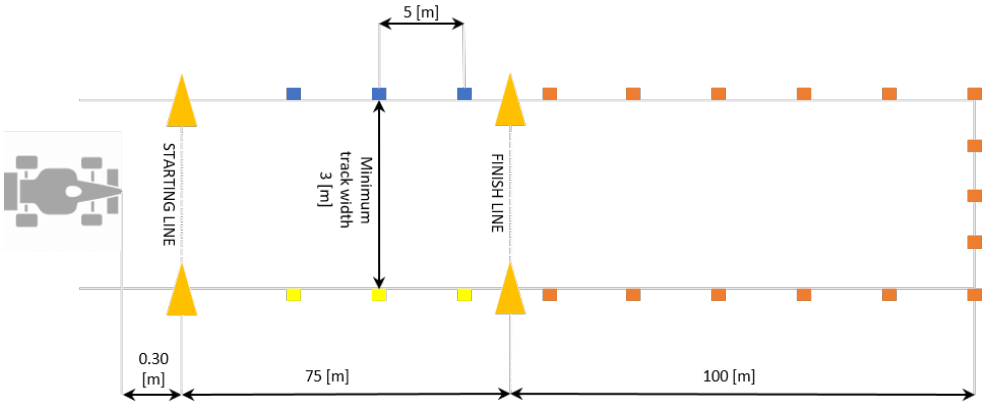



**Figure 2.2:** *Acceleration Procedure Track layout.*

(DQ). For Driverless Vehicles only, if the time set, including penalties, such as knocking over the cones or the entire base of the cone is pushed outside the box marked around it, DDO, Down or Out condition, 2 s for the acceleration event, D9.1.3 [39]) is less than $T_{max}$, additional points based on the following formula are given:

$$ACCELERATION\_SCORE = 71.5 \left( \frac{T_{\max}}{T_{\text{team}}} - 1 \right) \tag{2.1}$$

$T_{team}$ is team's best time including penalties;
$T_{max}$ is 2 times the time of the fastest vehicle including penalties.

## 2.2 Hardware Concept - SC19

The SquadraCorse Polito driverless racecar SC19 *Lucia* is an electric AWD racecar with a full aerodynamic package, lightweight design and high drivetrain efficiency, see Figure 2.3. In Table 2.2 are reported the main specifications of the SC19 racecar.
The vehicle Hardware Architecture can be divided into three main categories:

1. *Perception*: 3D LiDAR sensor(Velodyne VLP-16™) and a stereo camera sensor (ZED™) in order to sense the environment;

2. *Steering Actuator*: To autonomously follow the trajectory calculated on board by the AV system, the vehicle is equipped with a steering actuator, capable of

moving the steering rack with a brushless electric motor;

3. *Emergency Brake Sytem*: In case the AV system fails or vehicle dynamic instability occurs when race car is running, it may be necessary for safety reasons to use an emergency system that is activated autonomously or by pressing a remote button (RES) that generates pressure in the braking system lines of the car and therefore stop it in the conditions described above.

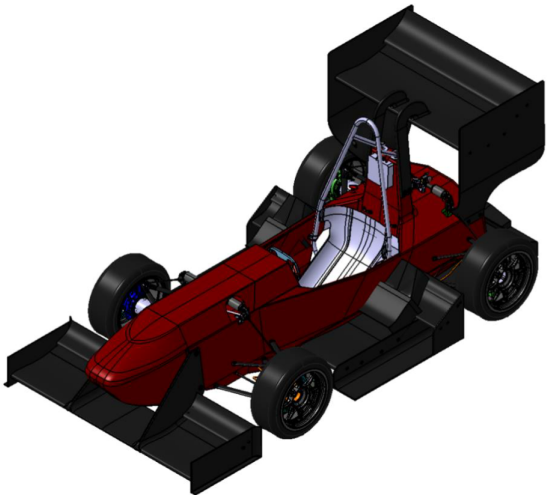


**Figure 2.3:** *CAD model of SC19 Lucia.*

### Perception

In order to make the vehicle race fully autonomously, sensors allowing for environment perception need to be added. As all the perception algorithms, decision making and control will have to be executed on board, also additional computational units have to be added. Trivially for security reasons, the Perception system is required to be robust and reliable, which calls for redundancy in the Perception pipeline. Therefore, two independent perception pipelines working in parallel were considered, in order to have a robust system perception architecture.

The *first* way to perceive the surrounding environment is the 3D LiDAR sensor placed in the middle of the front wing. LiDAR is categorized as an active remote sensing technology because it actively transmits an electromagnetic pulse of energy (in the

**Table 2.2:** *Technical specifications of SC 19 race car. *It has been considered the driver's weight.*

| **SC19 Technical Specifications** | |
|---|---|
| Parameter | Value |
| Mass* | 253 kg |
| Moment of Inertia about z-axis* | 95.81 $kgm^2$ |
| Vehicle wheelbase | 1,525 m |
| Overall length | 2,873 m |
| Front axle distance to CG | 0.839 m |
| Rear axle distance to CG | 0.686 m |
| Vehicle track width | 1.4 m |
| Overall width | 1.38 m |
| Height of CG* | 0.242 m |
| Wheel radius | 0.241 m |
| Maximum power (total vehicle) | 80 kW |
| Motors peak torque (total vehicle) | 84 Nm |
| Transmission ratio | 14.82 |
| Maximum energy stored (battery pack) | 6.29 kWh |
| Battery pack voltage | 558 V |

optical range) used to measure distance (Velodyne™ Lidar sensor). The choice of the LiDAR sensor over radars is based on its physical parameters like the horizontal and vertical resolution and fields-of-view (FoV). The vertical resolution appears to be the most important parameter since it limits the number of returns per cone and therefore the distance at which the cones can be perceived. In Table 2.3 are reported the specifications of Velodyne™ LiDAR sensor (Puck™) mounted on the SC19 race car [26].

The *second* way to perceive the surrounding environment is image-based. The stereo camera in question (ZED™ from STEREOLABS™) is passive. This reproduces the way human vision works. Using its two "eyes", the ZED™ creates a three-dimensional map of the scene by comparing the displacement of pixels between the left and right images. The ZED™ captures two synchronized left and right videos of a scene and outputs a full resolution side-by-side color video on USB 3.0. This color video is used by the ZED software on the host machine to create a depth map of the scene, track the camera position and build a 3D map of the area. The stereo camera is placed on the main roll hoop, above the driver's seat in the car; this offers the advantage that the occlusion among cones is reduced to a minimum and even the cones placed one behind

the other (in line of sight) can be perceived sufficiently well. In Table 2.4 are reported the specifications of STEREOLABS™ stereo camera sensor (ZED™) [38].

### Steering Actuator

The steering actuator is an electromechanical linear actuator with a brushless DC motor (Maxon™ motor EC 60 flat Φ 60 mm) whose rotor shaft is coupled with a screw shaft. A nut is engaged with the screw shaft and when the motor rotates, it drives the nut linearly according to the screw mechanism thus converting rotary motion into linear motion. As can be seen in Figure 2.4, the nut is fixed and rotating it using a belt drive it gets the linear actuation through screw. The movement is then transferred to an actuator link (yellow component in Figure 2.4); this is mounted under the steering rack supported by the same rack mountings and connected to the clevis joints on both sides (orange components in Figure 2.4). This actuator link slides through a low friction plain bearing in the rack mounting while translating in the Y axis. In Table 2.5 are reported the specifications of the Steering Actuator mounted on SC 19 prototype.

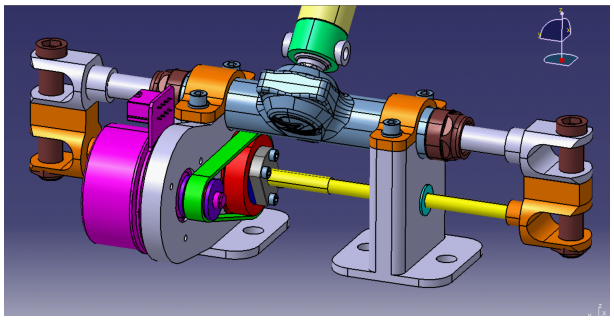


**Figure 2.4:** *CAD of the Steering Actuator assembly.*

### Emergency Brake System

The used autonomous racecar is capable of braking by recuperation through the motors. This generates a sufficient deceleration in order to race without a driver while not using the mechanical brakes actively. An Emergency Braking System (EBS) was installed behind the pedals as shown in Figure 2.5b. The EBS is a passive pneumohydraulic

**Table 2.3:** *Technical specifications of Velodyne™ VLP-16 installed on the SC 19 protype.*

| **Velodyne™ VLP-16 Technical Specifications** | |
|---|---|
| Sensor: | Time of flight distance measurement with calibrated reflectivity<br>16 channels<br>Measurement range 1 to 100 m<br>Accuracy +/- 3 cm (typical)<br>Duel Returns (strongest and last)<br>Field of view (vertical): 30 ° (+15 ° to -15 °)<br>Angular resolution (vertical): 2 °<br>Field of view (horizontal/azimuth): 360 °<br>Angular resolution (horizontal/azimuth): 0.1 °-0.4 °<br>Rotation rates: 5-20 Hz |
| Laser: | Class 1 – eye safe<br>903 nm wavelength (min/max is 896/910 nm)<br>Firing sequence repetition rate: 55.296 µs/18.2 kHz<br>Pulse duration: 6 ns<br>Maximum output energy: 31 Watts (0.19 mJ) |
| Mechanical/<br>Electrical/<br>Operational: | Power consumption: 8 W (typical)<br>Operating voltage: 9-32 VDC (with interface box and regulated power supply)<br>Weight: 830 g (without cabling)<br>Dimensions: 103 mm diameter x 72 mm height<br>Shock: 500m/s$\hat{2}$ amplitude, 11 ms duration<br>Vibration: 5 Hz to 2000 Hz, 3G rms<br>Environmental protection: IP67<br>Operating temperature -10 °C to +60 °C<br>Storage temperature -40 °C to +105 °C |
| Output: | Data output: $\sim$ 0.3 million points/s<br>100 Mbps Ethernet Connection<br>UDP packets containing:<br>Distances<br>Calibrated reflectivity<br>Rotation angles<br>Synchronized time stamps (µs resolution)<br>$GPRMC NMEA sentence from FPS receiver |

**Table 2.4:** *Technical specifications of ZED™ stereo camera sensor installed on the SC 19 protype.*

| **STEREOLABS™ ZED™ stereo camera Technical Specifications** | | | |
|---|---|---|---|
| Video Output: | Video Mode | Frames per second | Output Resolution (SxS) |
| | 2.2 K | 15 | 4416x1242 |
| | 1080p | 30 | 3840x1080 |
| | 720p | 60 | 2560x720 |
| | WVGA | 100 | 1344x376 |
| | Video Recording: Native resolution video encoding in H.264, H.265, Lossless formats (on host) | | |
| | Video Streaming: Stream anywhere over IP | | |
| Deph: | Depth Resolution: Native video resolution (in Ultra mode) | | |
| | Depth FPS: Up to 100Hz | | |
| | Depth Range: 0.3 - 25 m (0.98 to 82 ft) | | |
| | Depth FoV: 90° (H) x 60° (V) x 100° (D) max. | | |
| | Technology: Stereo Depth Sensing | | |
| Motion: | Lens Type: 6-elements all-glass dual lens | | |
| | Aperture: f/2.0 | | |
| | Field of View: 90° (H) x 60° (V) x 100° (D) max. | | |
| Image Sensors: | Sensor Resolution: Dual 4M pixels sensors with large 2-micron pixels | | |
| | Sensor Size: 1/3” backside illumination sensors with high low-light sensitivity | | |
| | Camera Controls: Adjust Resolution, Frame rate, Brightness, Contrast, Saturation, Gamma, Sharpness, Exposure and White Balance | | |
| | Sensor Format: Native 16:9 format for a greater horizontal field of view | | |
| | Shutter Sync: Electronic Synchronized Rolling Shutter | | |
| Connectivity: | Connector: USB 3.0 port with 1.5 m integrated cable | | |
| | Power: Power via USB 5 V / 380 mA | | |
| | Mounting Options: 1/4”-20 UNC thread mount | | |
| | Operating Temperature: 0 °C to +45 °C | | |
| Size and Weight: | Dimensions: 175 x 30 x 33 mm | | |
| | Weight: 135 g | | |

**Table 2.5:** *Technical specifications of the Steering Actuator system.*

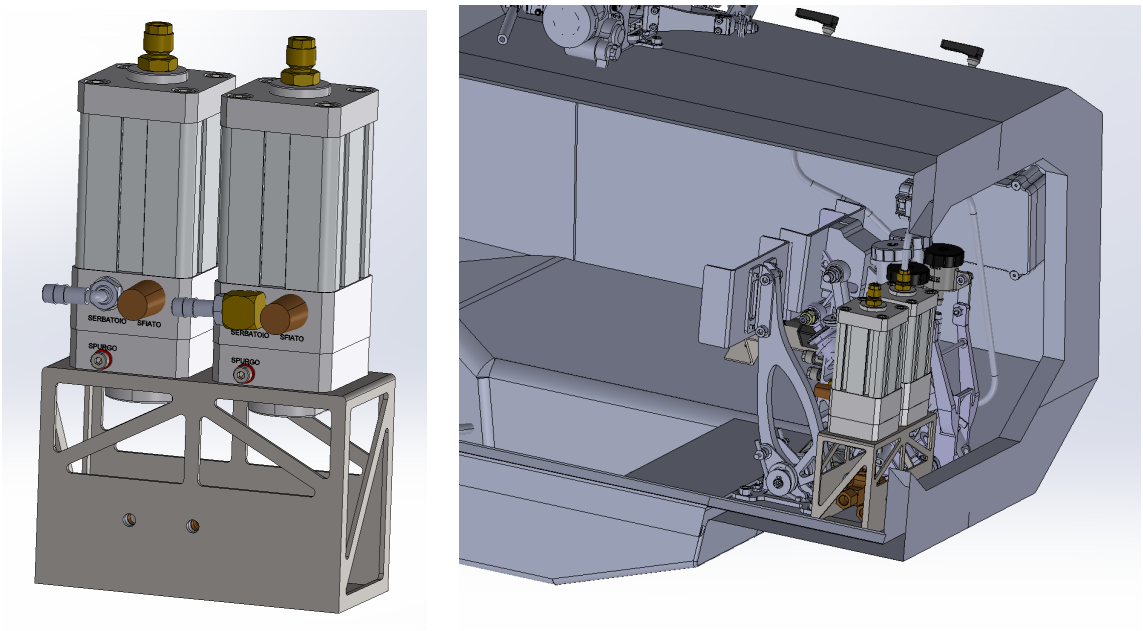
| **Steering Actuator Technical Specifications** | |
|---|---|
| Rack lenght: | 263 mm |
| Actuation required lenght: | 45 mm |
| Actuation required speed from full left to full right: | 45 mm/s |
| Reversibility: mandatory for regulations; both human and electrically driven | |
| Supply voltage: | 12 V |
| Weight: | 1.2 kg |
| Torque required at ball screw: | 397 mNm |
| Gear Ratio (Pulley): | 2 |
| Selected Motor torque at 12 V: | 261 mNm |

**Table 2.6:** *Technical specifications of the EBS.*

| **EBS Technical Specifications** | |
|---|---|
| Required pressure Brake lines pressure: | 40 bar |
| Required pressure for compressed air: | 10 bar |
| Actuation time: | 130 ms |
| Weight: | 2.3 kg |
| Pressure Loss at each actuation: | $\sim$ 2 bar |

system that converts the air pressure contained in high pressure tanks (HPA) into braking torque for the vehicle's brake lines, thanks to pressure boosters, Figure 2.5a. Electrical powerloss at EBS must lead to a direct emergency brake maneuver. (DV 3.1.3) This system is only able to either fully brake or release the mechanical brakes and is used only in case of emergency. It can be triggered from either a Remote Emergency System (RES) or from an on-board computer. The system reaction time (the time between entering the triggered state and the start of the deceleration) must not exceed 200 ms.(DV 3.3.1) In Table 2.6 are reported the specifications of the Steering Actuator mounted on SC 19 prototype.

In Figure 2.6, the position of the various components monted on the SC19 race car is shown.



**(a)** *Details of EBS: Intensifiers and OR valves.* **(b)** *EBS behind the brake pedal of SC19 race car.*

**Figure 2.5:** *Emergency Brake system (EBS).*

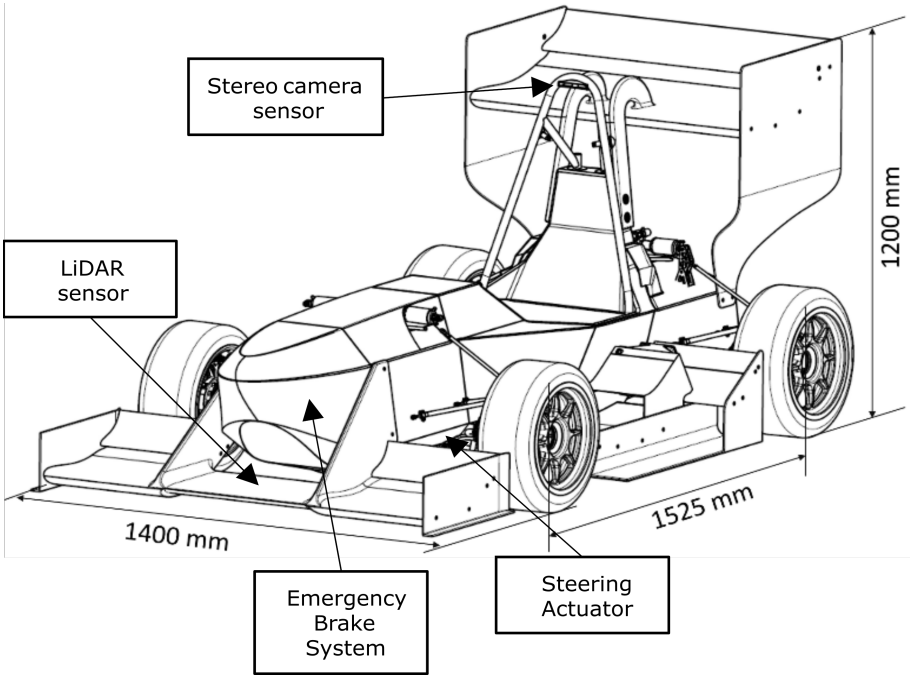



**Figure 2.6:** *Hardware components position on SC 19 race car.*

Chapter 3

# Path Planning

Path planning is a merely geometric matter, because it is defined as the generation of a geometric path, with no mention of any specified time law [16]. The definition of the path planning problem is very straightforward: "find a collision-free motion between an initial (*Start*) and a final configuration (*Goal*) within a specified environment". The simplest situation is when the path is to be planned in a static and known environment; however, more generally, the path planning problem can be formulated for any robotic system subject to kinematic constraints, in a dynamic and unknown environment. Much work can be found in robotic literature, dealing with path planning. The first definitions and algorithms date back to the 1970's. Jean-Cloude Latombe published a complete overview of the path planning techniques [28] in 1991. Then others overview of many path planning techniques can be found in the classic book "Principles of robot motion: theory, algorithms, and implementation" [8] or in more recent book "Motion Planning" [22].

Some basic definitions are needed to introduce the path planning problem, namely:

1. The configuration space ($\mathcal{C}$-space);
2. The space of free configurations ($\mathcal{C}$-free);
3. the obstacles' representation in the $\mathcal{C}$-space ($\mathcal{C}$-obs).

The configuration space is the space of all possible robot configurations, where a configuration q is the specification of position and orientation of the robot A with respect

to a fixed reference frame FW. Referring to Figure 3.1, the $\mathcal{C}$-space of the robot A is $\mathcal{R}^3$, since the configuration of A is specified by the origin of FA with respect to FW, and by its orientation. The $\mathcal{C}$-obs is given by the image of the obstacles in the $\mathcal{C}$-space, and the $\mathcal{C}$-free is defined as ($\mathcal{C}$-space - $\mathcal{C}$-obs).

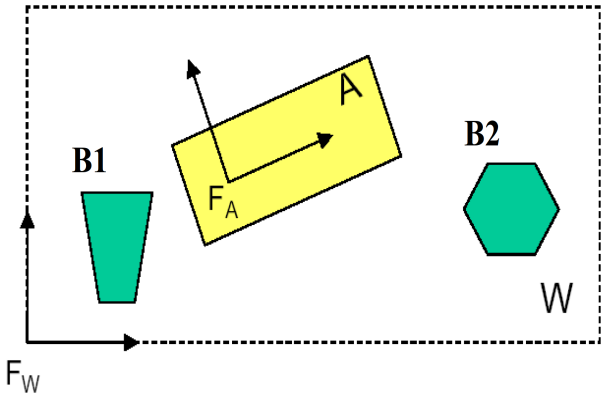



**Figure 3.1:** *Mobile robot in a 2-dimensional space with obstacles.*

## 3.1 State of Art

Path planning algorithms are usually divided in three categories, according to the methodologies used to generate the geometric path, namely:

1. Roadmap techniques;
2. Cell decomposition algorithms;
3. Artificial potential methods;
4. Alternative approaches to path planning.

### 3.1.1 Roadmap techniques

The roadmap techniques are based upon the reduction of the N-dimensional configuration space to a set of one-dimensional paths to search, possibly on a graph. In other words, this approach maps the free space connectivity into a system of one-dimensional

curves (the roadmap) in the C-free space or in its closure [16]. The roadmap RM thus, is a union of curves such that for all start and goal points in $\mathcal{C}$-free that can be connected by a path:

1. *Accessibility*: There is a path from $q_{start} \in \mathcal{C}$-free to some q' $\in$ RM;
2. *Departability*: There is a path from some q" $\in$ RM to $q_{goal} \in \mathcal{C}$-free;
3. *Connectivity*: There exists a path in RM between q' and q";
4. *One dimensional*.

One *Method* related to this technique is the Visibility Graph. It is defined for polygonal obstacles and every node correspond to the vertices of the obstacles. Nodes are connected if they are already connected by an edge on an obstacle or the line segment joining them is in $\mathcal{C}$-free space. This method guarantees always a path inside the roadmap and above all this path is the *shortest*. The concept of visibility graph, which represents a milestone in the literature related to path planning, was introduced by Lozano-Pérez in 1979 [32]. In Figure 3.2, it is reported an *example* of the *Visibility Graph* method. Another kind of roadmap algorithms are those based on *Voronoi di-*

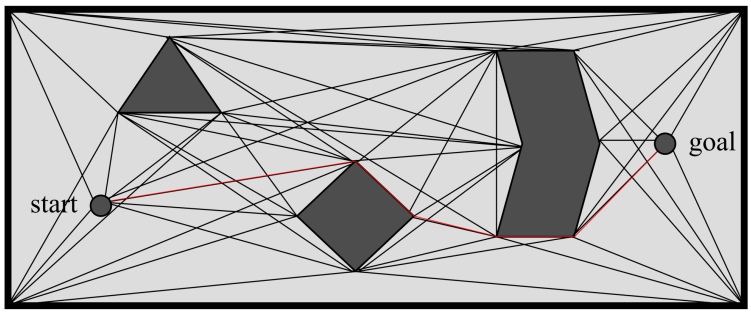



**Figure 3.2:** *Path planning using Visibility Graph method.*

*agrams*. These are defined as a way of partitioning a plane with \ points into *convex polygon* such that each polygon contains exactly one generating point and every point in a given polygon is closer to its generating point than to any other.
This approach is dual to that based on the Visibility Graph method, because the Voronoi diagrams enable to obtain a path lying at the maximum distance from the obstacles, whereas the Visibility Graph, as can be seen in Figure 3.2, generates a path

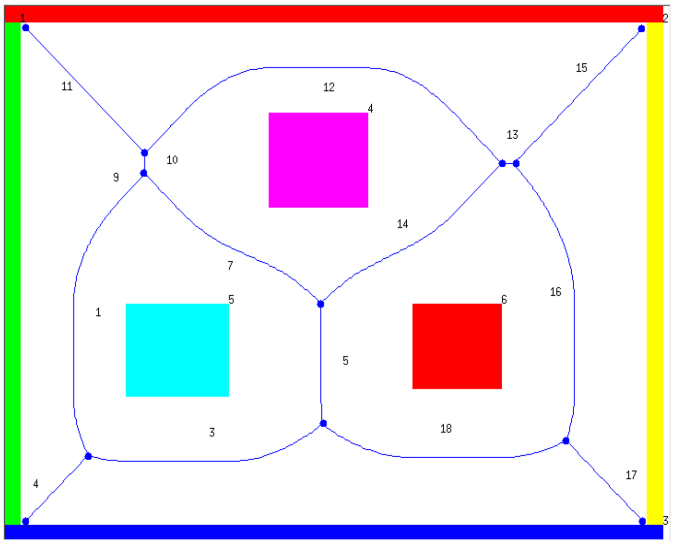



**Figure 3.3:** *Path planning using Voronoi diagrams.*

that passes as close as possible to the obstacle vertices. In Figure 3.3, some path generated by Voronoi diagrams are reported. The squares represent the obstacles while the blue lines are the set of points equidistant from at least two obstacles. Examples of path planning algorithms may be found in 1988 by Canny & Donald [7], in 1989 by Takahashi & Schilling [40] and in 2011 by Garrido et. al. [15].

### 3.1.2 Cell decomposition methods

The idea of these methods is to find obstacles-free regions, and build a graph of adjacency for them. In general two categories of cell decomposition algorithms are existed;

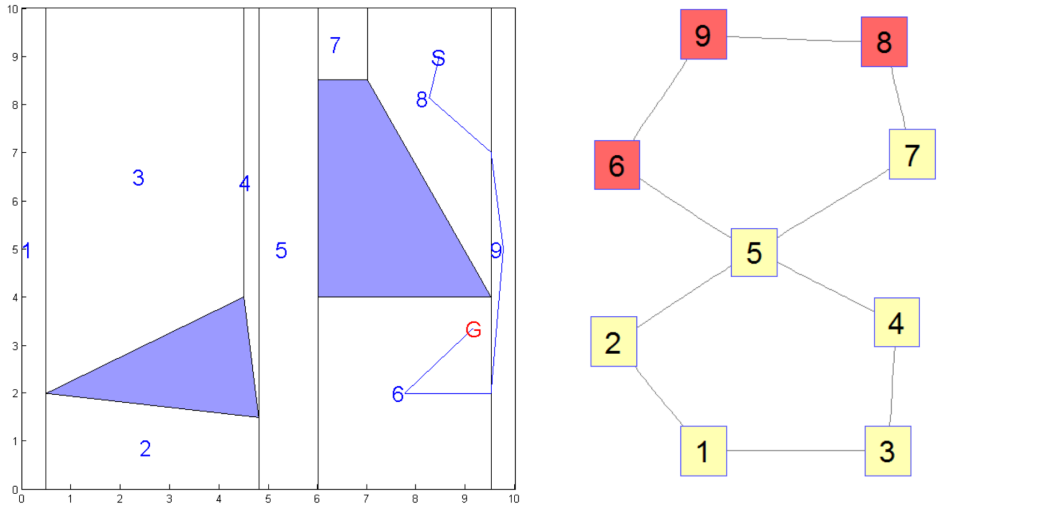
1. The exact cell decomposition methods;

2. The approximation methods.

#### Exact cell decomposition

The trapezoidal decomposition method or vertical cell decomposition decomposes the free space into trapezoidal and triangular cells. It draws parallel segments from each polygon's vertex in the workspace to the exterior boundary. The generated cells form the nodes of the so-called *connectivity graph*. The adjacent nodes in the workspace are linked to form the edges in the connectivity graph [1], [2]. The path in this graph

corresponds to sequence of striped free cells. When planning query is establish, the




**(a)** *Trapezoidal cell decomposition with vertical free cells graph.*

**(b)** *Graph of adjacency which correspoinding to path between cells.*

**Figure 3.4:** *Trapezoidal cell decomposition method.*

planner finds the start and goal cells, then it searches for a path between these two cells, if a path is found the planner connect the start and goal locations through the free cells on that path [3]. Again, the path planning problem is turned into a graph searching problem, anc can therefore be solved using graph-searching techniques. Figure 3.4 illustrates the procedure described above, named *Exact* cell decomposition technique, because the union of the cell represents exactly the free space [16].

### Cell decomposition approximation

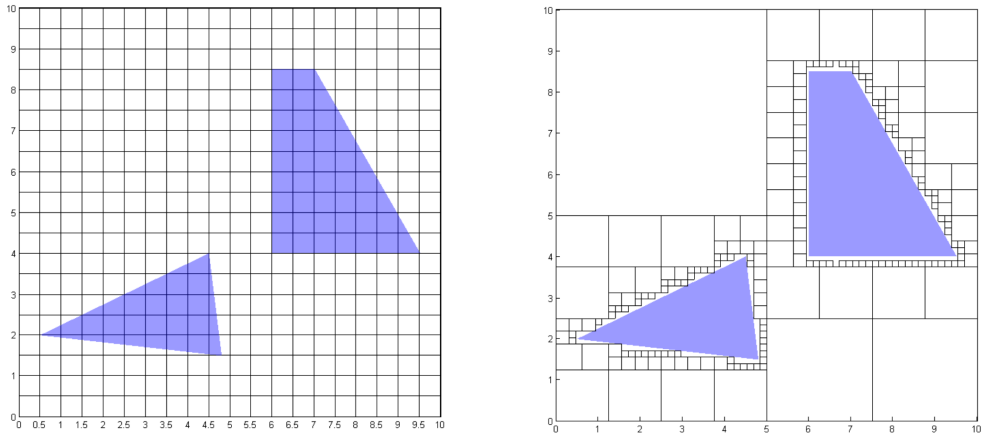
In some cases, an exact computation of the free-space is not possible or convenient. The approximation methods infact were proposed due to high computation and geometric calculation which are required by exact cell decomposition. The most forward approximate cell decomposition method is voxel grid. It uses regular voxel grid or pixel grid, as been reported in Figure 3.5a. It excludes the cells on bostacle areas and builds a graph of adjacency for cells on free area. This method is efficient for low dimensions space. However, it generates large number of cells. Another improvement for approximate cell decomposition was by using quad-tree decomposition, Figure 3.5b. This approach uses a recursive method. It recursively subdivides the cells until one of

the following scenarios occurs:

1. Each cell lies completely either in a $\mathcal{C}$-free space or in the $\mathcal{C}$-obs region;

2. an arbitrary limit resolution is reached.

Once a cell fulfils one of these criteria, it stops decomposing. After decomposition steps, the free path is found by following the adjacent free cells.



**(a)** *Voxel approximation method.* **(b)** *Quad-tree approximation method.*

**Figure 3.5:** *Cell decomposition approximation method.*

### 3.1.3 Artifical potential methods

These methods use artificial potential fields applied to the obstacles and goal positions and use the resulting field to influence the path of the robot which is subject to this potential [41]. Although not as thorough as the graph searching techniques, the speed of the algorithms and the easy extension to higher dimensions make them an excellent alternative to the graph searching techniques. The basic idea is to consider the robot in the configuration space as a moving point subject to a potential field generated by the goal configuration and the obstacles in the $\mathcal{C}$-space. The target configuration (*goal*) produces an *attractive* potential, while the obstacles generate a *repulsive* potential. The sum of these two contributions is the *total* potential, which can be seen as an artificial force applied to the robot, aimed at approaching the goal and avoiding the obstacles. Thus, given any configuration during the robot motion, the next configuration can

be determined by the direction of the artificial force to which the robot is subjected. The artificial potential method was originally conceived by Khatib in 1986 [24] and further developed by Volpe in 1988 [25]. However, the major problem with these potential field methods is that they are subject to local minima. Since the planner tends toward lower potential areas, it can reach a state of equilibrium, or a potential basin, and becomes trapped. Koditschek [27] presents a rigorous description of the topological considerations of the potential fields, introducing potential functions, also called *navigation function*, that have only one global minimum and no local minima. Another approach to solve the path planning problem is found in 1991 by Barraquand & Latombe [4], where a special kind of planners, named RPP (Random Path Planners), is proposed: local minima are avoided by combining the concepts of artificial potential field with random search techniques.

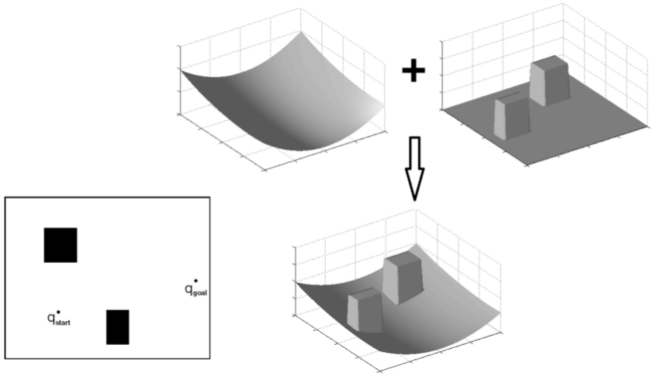


**Figure 3.6:** *Path planning using Artifical potential method.*

### 3.1.4 Alternative approaches to path planning

A possible alternative approach is given by the Probabilistic RoadMap Planners (PRM). It is a technique which employs probabilistic algorithms, such as random sampling, to build the roadmap. The basic idea is to consider a graph where the nodes are given by a set of random configurations in the $\mathcal{C}$-free. A local planner can then try to connect these configurations by means of a path: if a path is found, a new node is added to the graph. In this way the graph reflects the connectivity of the $\mathcal{C}$-free. In order to find a path between two configurations, these configurations are added to the graph, then a

graph search is performed in order to find a feasible path.
There are some examples of path planners that take into account kinematic and dynamic constraints of the robot, in addition to the pure geometric problem of obstacle avoidance [11], [13]. Kinodynamic and nonholonomic motion planning can be handled by the *Rapidly-exploring Random Tree* (RRT) method [29]. This method allows to search non-convex high-dimensional spaces by randomly building a space-filling tree.

## 3.2 Methodology

The method used in this project thesis is an on-line approach based on a modified RRT algorithm for a non-holonomic car-like mobile robot. The RRT algorithm is based on incremental construction of search tree that attempts to rapidly and uniformly explore obstacle-free segment in configuration space. After search tree is successfully created, a simple search among the branches of the tree can result in collision-free path between any two points in robot's environment. Due to differential constraints of non-holonomic car-like mobile robot, the set of potential configurations that robot can reach from certain state is reduced. For that reason basic form of RRT algorithm is unusable for path planning in case of non-holonomic robot and it needs to be modified. For this reason, *Dubins* curves are selected for representative model of our car-like robot with differential constraints. Difference between basic and modified RRT algorithm based on *Dubins* curves is in the way how branches in search tree are formed. In basic RRT, branches are made of straight lines, while modified algorithm uses *Dubins* curves to satisfy differential constraints.

#### Hardware layout used for experimental tests

The *Local path planner* must work using the data generated by the LiDAR and Stereo camera sensor as input. To do this, the hardware diagram shown in Figure 3.7 was created and mounted in the car. A lithium-ion battery powers an NVIDIA™ Jetson AGX Xavier at which the perception sensors (LiDAR and Stereo camera) are connected. The NVIDIA OS is the Ubuntu 18.4 and the ROS packages installed. ROS is the

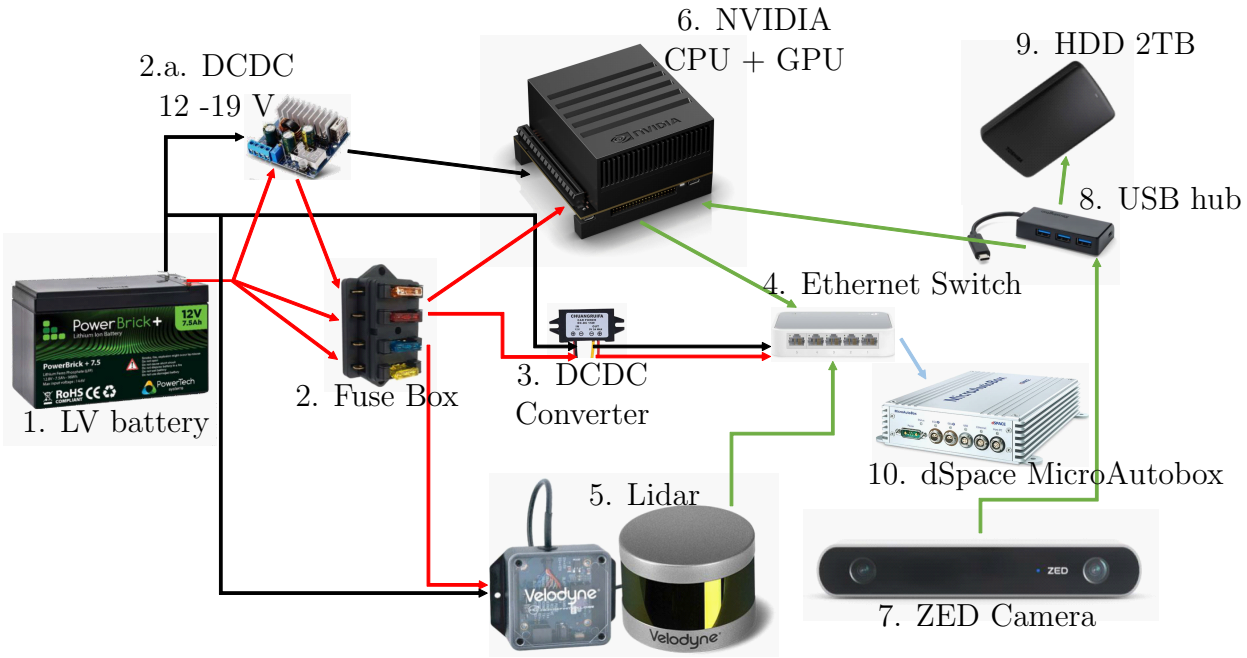
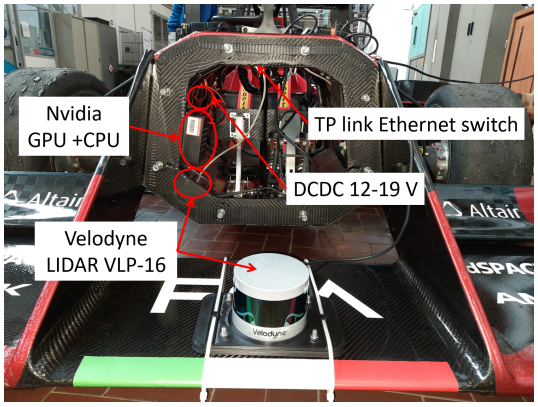


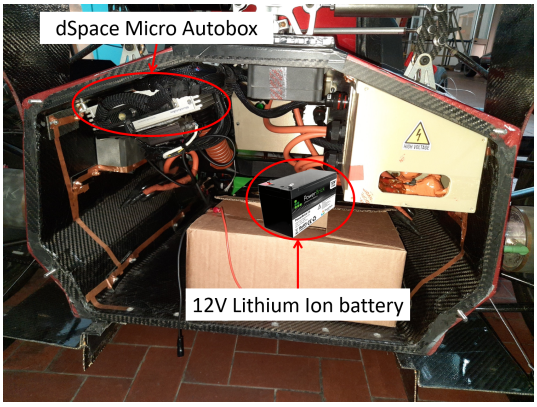

**Figure 3.7:** *Hardware setup onboard the racecar for data acquisition during experimental tests.*

only software platform able to manage the signals in real time coming from different sensors, emulate the robot (i.e our racecar), route TCP / IP connections, manage IP networks, implement AI-based algorithms and create SLAM (Simulataneous Localization and Mapping) for autonomous driving.

The dSpace™ MicroAutobox will then be able to manage the data processed by the NVIDIA™ micro CPU and give them as input to the *Path planning* model built on Simulink™. In the early phase of definition and construction of the *Path planning* model, since the integration with the dSpace™ Microautobox was not carried out and since the steering actuator and the EBS Emergency Braking System had not yet been installed in the racecar, it was decided to acquire the data generated by the LiDAR and the Stereo camera sensor by driving the racecar in manual mode around different scenarios that recreate the *Acceleration* and *Trackdrive* mission. In this way, through the hardware setup shown in Figure 3.7, it is possible to save the data generated by the perception sensors in an external Hard Disk (component 9 in Figure 3.7). In this way, the effectiveness, efficiency, robustness and working conditions of the model are tested considering data congruent with the future *mission* the racecar has to accomplish. The data thus generated by the LiDAR and stereocamera sensor were saved in an external Hard Disk (HD) for the entire duration of the acquisition. These data,




**(a)** *View of the front of the racecar with some components in sight.*




**(b)** *View of the rear of the racecar with some components in sight.*

**Figure 3.8:** *Positioning in the racecar of some components of the Hardware setup shown in Figure 3.7.*

extracted from the HD, are then post processed by simulating the real racecar working condition in which, for example, the data coming from the LiDAR sensor are processed by the *Local path planner* in the *Path planning* model thus generating the trajectory the vehicle must follow for the *Acceleration* mission.
Figure 3.7 shows the Hardware Layout that allows to save the data from the LiDAR and Stereo camera sensor to an external hard disk (HHD 2TB component 9). Figure 3.8a and Figure 3.8b show the installation and positioning of the components in the racecar.

### Methodology Framework

The next sections will be divided into two parts in order to better understand the methodology that led to the definition of the *Local Path planner*. The two parts also follow chronologically the steps that have been taken to achieve the thesis project objectives. These steps are:

1. Automatic goal generation for each frame;

2. Generation of the modified RRT algorithm with Dubins curves.

Secondly it is explained the structure of the SIMULINK™ model and a detailed description of the controller and vehicle model coupled with the *Local Path planner*.

### 3.2.1 Automatic generation of the goal

In order to obtain a feasible trajectory between initial and final configuration for non-holonomic car-robot using modified RRT algorithm, it needs to define the goal or the final configuration so as to be able to generate the *Dubins* curves.
As already mentioned in Paragraph 2.1, it has been considered the *Acceleration* mission as Problem Statement. During the mission, the vehicle has to go straight for 75 m as fast as possible, Figure 2.2. Therefore there is no need to use the ZED™ stereo camera sensor to define the colors of the cones since those on the left side of the track will always have blue color as well as the yellow ones will always be on the right side. For this reason, it was decided to use only the data coming from the Velodyne™ VLP-16 LiDAR sensor with a frequency of acquisition of 10 Hz in order to define the boundary of the *Acceleration* mission track layout.
Thus, every 0.1 s the Path Planner receives the cartesian coordinates of the cones on a 2-D map referenced with respect to the position of the car. To eliminate the noise coming from the LiDAR sensor, a filtering action is carried out before these data enter as input in the *Path Planner* block. The filtering action is composed by three condition:

1. Take only points that have positive abscissa;
2. Take points that have maximum distance with respect to the vehicle position within 20 m along the x axis;
3. Take points that have maximum distance with respect to the vehicle position within the interval of -5 m and 5 m along the y axis.

In Figure 3.9 it is reported the points sensed by the LiDAR sensor on a 2-D map for one single frame. Red points represent the filtering process, in other words, the points that satisfy the three conditions written above. After the extrapolation step above, four dummy points are added. These points are:

1. Two points on the side of the vehicle with cartesian coordinates ✳ (0,1.51), (0.1,-1.51);
2. Two points on the side of the vehicle but positioned in front of it with cartesian coordinates ✳ (2,2.1), ✳ (2.05, -2).

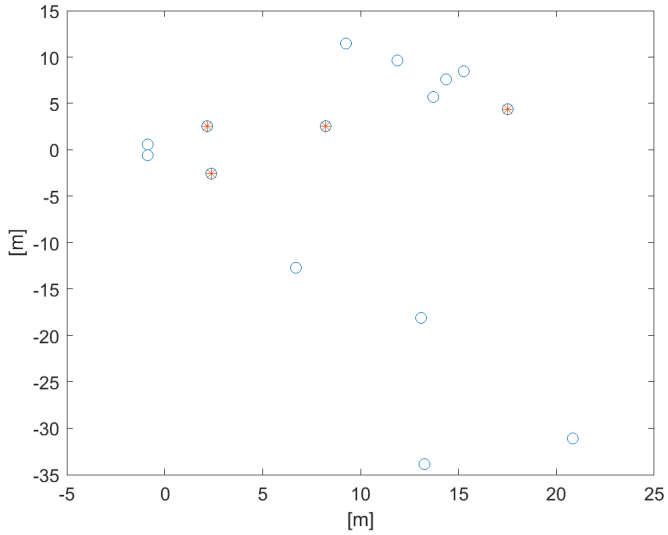



**Figure 3.9:** *Filtering phase of the relevant points.*

In Figure 3.10 the addition of the four points defined above is reported. Extrapolating

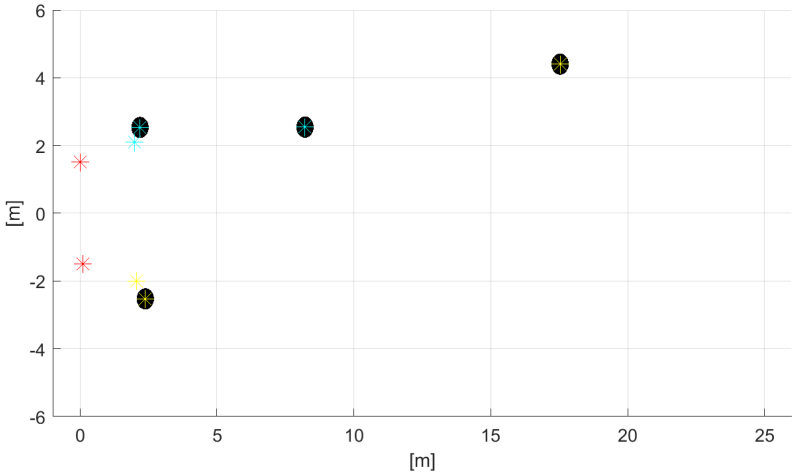



**Figure 3.10:** *Addition of fictitious points to the frame.*

points of about 2000 frames in various track layout such as straight segments, left or right curves, it was found necessary to add further fictitious points since the number of points extrapolated according to the above criteria are in 50% of cases insufficient to characterize the layout of the track and build the Delaunay Triangulation, which will be the next step for the automatic generation of the *goal*.

### Delaunay Triangulation

Since the vehicle drives between the cones, these can be used to discretize the space within which the car moves. Thus, the X-Y space is discretized by performing a *Delaunay Triangulation* [10].

In computational geometry, a *Delaunay Triangulation* for a given set P of discrete points in a plane is a triangulation DT(P) such that no points in P is inside the circumcircle of any triangle in DT(P). In this way, the vertices of the triangulation are the cone observations and the triangles are chosen such that the minimum internal angle of each triangle is maximized. For four or more points on the same circle (e.g., the vertices of a rectangle) the *Delaunay triangulation* is not unique: each of the two possible triangulations that split the quadrangle into two triangles satisfies the "Delaunay condition", i.e., the requirement that the circumcircles of all triangles have empty interiors. This is the reason why the cartesian coordinates of the fictitious points defined in Paragraph 3.2.1 are asymmetrical respect to the x axis so as to obtain a unique *Delaunay Triangulation*.

Basically for the generation of the Delaunay Triangulation is used an*iterative search* technique. Since one and only one circumference passes through three non-aligned points, for each random triplet of points taken from the array of points that make up the frame, formed by the filtered points coming from the LiDAR sensor and by the four fictitious points added previously, it is calculated the circumference passing by these three points. if the created circle does not circumscribe any point belonging to the frame, then the triplet of points is saved. This triplet of points in fact defines the vertices of the triangle that will make up the *Delaunay Triangulation*. The iterative process is repeated until all possible combinations of points are investigated.

The Procedure of the *Delaunay Triangulation* algorithm (1) is presented below; to do so, it has been used the MATLAB™ software. The output obtained from this algorithm is the so called *Connectivity List* matrix. It is a matrix of size *mtri*-by-*nv*, where *mtri* is the number of triangles and *nv* is the number of vertices. Each row specifies a triangle defined by vertex IDs - the row numbers of the Points matrix, size *p*-by-*2*, where *p* is the number of points in a frame and *2* represents the x y cartesian coordinates.

In Figure 3.11 it is represented the *Delaunay Triangulation* construction and the vertex

**Algorithm 1** Delaunay Triangulation algorithm

```
1: procedure DT CONSTRUCTION(x_coord, y_coord)
2:     for i ← 1, size(x_coord) do
3:         for j ← 1, size(x_coord) do
4:             for k ← 1, size(x_coord) do
5:                 if (i ≠ j) AND (i ≠ k) AND (j ≠ k) then
6:                     points = [y_coord(i), x_coord(i); y_coord(j), x_coord(j); y_coord(k), x_coord(k)];
7:                     [R, xcyc] = circumference(points);
8:                     x_check[i, j, k] = [ ];
9:                     y_check[i, j, k] = [ ];
10:                    x_p = xcyc(1) + R * sin(t);
11:                    x_p = xcyc(2) + R * cos(t);
12:                    in = inpolygon(y_check, x_check, x_p, y_p);
13:                    if (sum(in) == 0) then
14:                        CList = [i, j, k];
15:                    end if
16:                end if
17:            end for
18:        end for
19:    end for
20: end procedure
```

IDs. Table 3.1 represesents the above written *Connecitivity List* matrix, where rows represents the number of triangles while columns correspond to the vertecies of the triangles. The generation of the *Delaunay Triangulation* aims to discretize the X-Y space and find the *midpoints* between the cones in order to generate a trajectory that has as initial configuration the point where the racecar is located and as final configuration the Goal.

After having generated the *Delaunay triangulation*, the automatic generation of the Goal for each frame takes place in 3 steps:

1. *PotPoints*: Definition of potential points;
2. *PotGoals*: Definition of potenital goals;
3. *finalGoal*: Definition of the final Goal;

### Potential Points - *PotPoints*

Every 0.1 s the path planner will receive, as input from the Lidar sensor, the data containing the Cartesian coordinates of the perceived cones. Therefore, every 0.1 s the

**Table 3.1:** *Connecitivity List matrix of the Delaunay Triangulation shown in Figure 3.11.*

| Connectivity List Matrix | | |
|---|---|---|
| 1 | 3 | 4 |
| 1 | 3 | 8 |
| 1 | 6 | 8 |
| 2 | 3 | 4 |
| 2 | 3 | 7 |
| 2 | 5 | 7 |
| 3 | 7 | 8 |
| 5 | 6 | 7 |
| 6 | 7 | 8 |

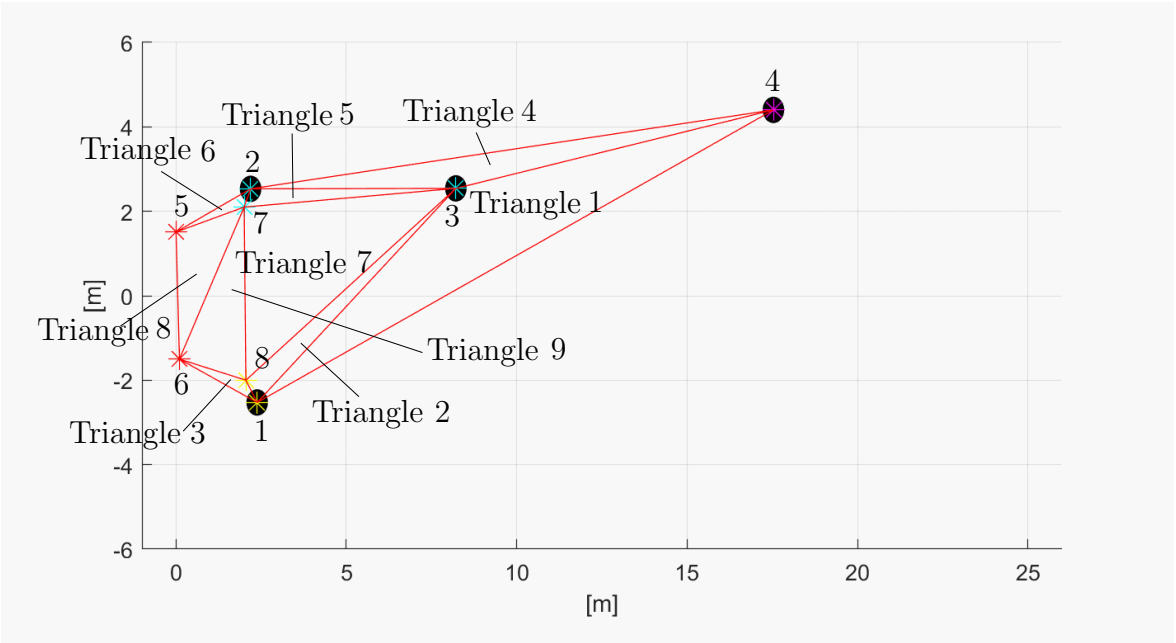



**Figure 3.11:** *Delaunay Triangulation construction. Vertex IDs and triangles are reported.*

trajectory planning algorithm must be able to automatically calculate the goal in order to define a trajectory for each frame. To do so, the first step is to calculate the *midpoints* of the edges of the triangles after the *Delaunay triangulation* (DT) construction is performed.

Looking at the triangles generated after the DT, see Figure 3.11, it is clear that it is not necessary to calculate the *midpoints* of all the edges of the triangles but rather to concentrate the attention on the edges that can potentially be found between the carriageway defined by the cones, so as to calculate the *midpoints* of our interest. With reference to the Table 3.1, two equal values (vertex IDs) belonging to different rows (Triangles) of the *Connectivity List* matrix means two triangles are adjacent and so they share the same edge. Our aim is to generate a trajectory in the middle of the carriageway defined by the cones and so the probability of calculating *midpoints* of edges adjacent to two triangles with blue and yellow cones as extremes is very high. For this reason, in this phase each row of the *Connectivity List* matrix is analyzed in order to extrapolate which elements are common to each row by comparing them. The common elements are thus saved in a matrix $A$ which will have as rows the number of edges of two adjacent triangles and as columns the vertex IDs that define the extremes of these edges. Referring to the Figure 3.1, the matrix $A$ reported in Table 3.2 is found. A second filtering operation is performed on this matrix in order to eliminate the rows that have two equal elements, meaning only one vertex is in common between two triangles. Then the vertex IDs coming from the reworked $A$ matrix, are used to extrapolate, from the frame, the segments from which we want to calculate the *midpoints*. These are then collected in the so called *PotPoints* array. Below it is reported the procedure to genenerate this array, Algorithm 2 and in Figure 3.12, the *PotPoints* are graphically reported.

**Table 3.2:** *A matrix from potential points Procedure.*

| **A matrix** | |
|---|---|
| 1 | 3 |
| 1 | 1 |
| 3 | 4 |
| 3 | 3 |
| 1 | 8 |
| 3 | 3 |
| 3 | 8 |
| 8 | 8 |
| .. | .. |

**Algorithm 2** Potential Points calculation

**procedure** POTPOINTS($CList, obs$)
  $dim = size(CList, 1)$;
  **for** $i \leftarrow 1, (dim - 1)$ **do**
    **for** $j \leftarrow 1, (dim - 1)$ **do**
      **if** $isempty(intersect(CList(i,:), CList(j+1,:))) == 1$ **then**
        $A = zeros(1, 2)$;
      **else**
        **if** $size((intersect(CList(i,:), CList(j+1,:))), 2) == 1$ **then**
          $A = [(intersect(CList(i,:), CList(j+1,:))), (intersect(CList(i,:), CList(j+1,:)))]$;
        **else** $A = intersect(CList(i,:), CList(j+1,:))$;
        **end if**
      **end if**
    **end for**
    $PotPoints = [(obs(A(i,1),1) + obs(A(i,2),1))/2, (obs(A(i,1),2) + obs(A(i,2),2))/2]$;
  **end for**
**end procedure**

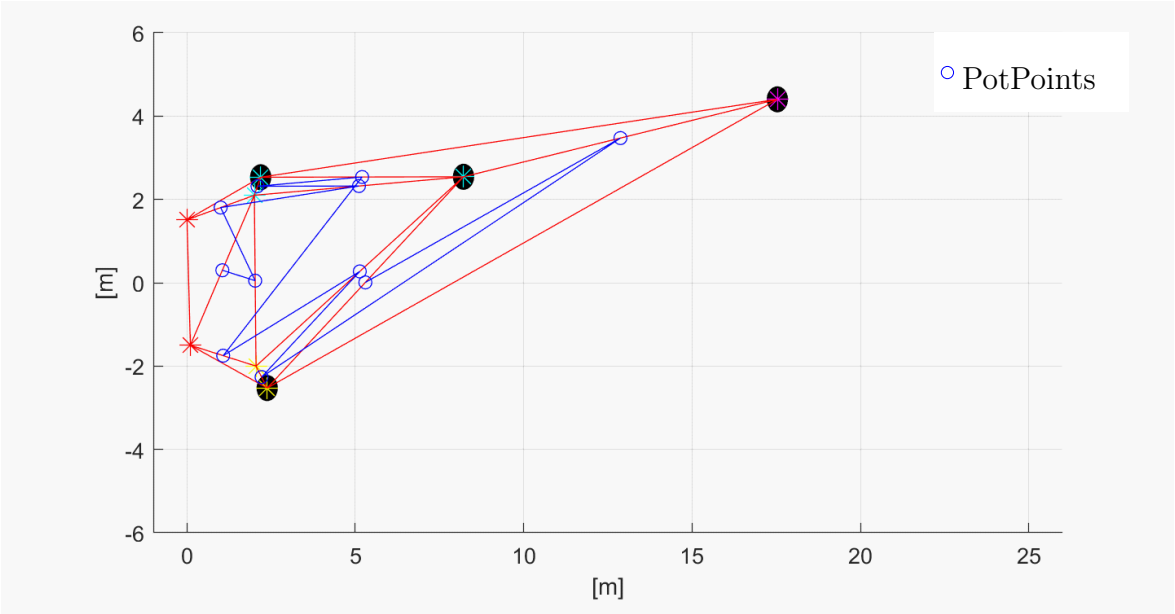



**Figure 3.12:** *Calculation of Potential Points (PotPoints).*

### Potential Goals - *PotGoals*

In this second step, from the array that composes the *PotPoints*, those *midpoints* that lie in the middle of the carriageway, located between a yellow and a blue cone are extrapolated. Considering that the data coming from the LiDAR sensor provide only the position of the cones with Cartesian coordinates with respect to the relative position of the vehicle, and not the information related to the color of the cones, it is necessary to find an alternative method to select the *midpoints* useful for the definition of the goal and eliminate those that are on the side of the carriageway and therefore not useful for our purpose.

To extrapolate the midpoints in the middle of the carriageway, the distance along the y-axis between the initial configuration of the vehicle, which for each frame corresponds to the $origin = [x, y] = [0, 0]$, and all the *midpoints* found before is calculated. Then this distance along the y-axis is set lower than a determined value, called *rd_v_potGoal* in order to select the midpoints in the middle of the carriageway defined by the cones and discharge those on the side of track.

Comparing numerous frames, it was noted that the *rd_v_potGoal* value must vary according to the structure of the frame, therefore depending on whether the LiDAR sensor senses a straight or a curve. This is done in order to minimize the error of

selecting *midpoints* which are on the side of the circuit. Therefore a code has been implemented that allows to calculate the area of a rectangle that has as height the maximum and minimum of the ordinate of the point comparing all the obstacles belonging to the specific frame and lenght equal to 1 m. In order to set the proper value for the *rd_v_potGoal* variable, it has calculated the rectangular area for each frame and on the base of the average value of the rectagular area over about 2300 frames, it has choosen the proper value for *rd_v_potGoal*, Figure 3.13.

With reference to the average rectangular area value found before and to the rules

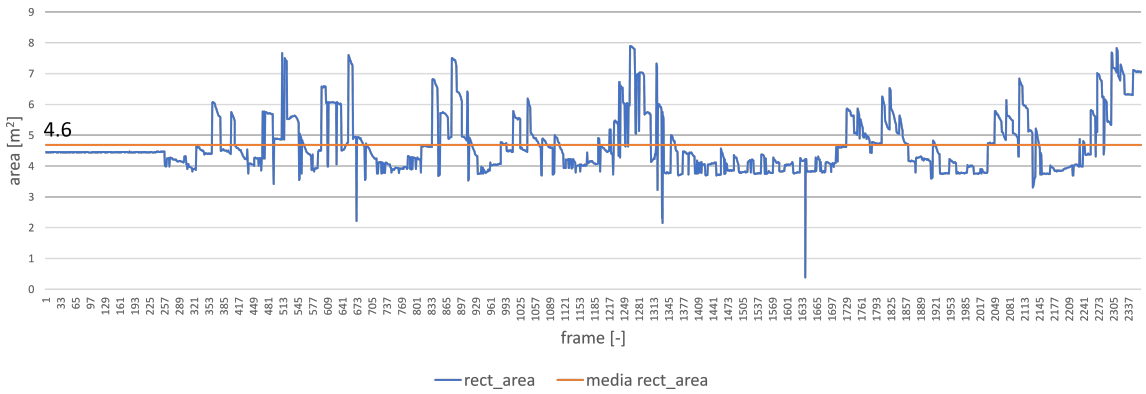



**Figure 3.13:** *Rectangular area value for about 2300 frame. Average rectangular area value about 4.6* $m^2$.

concerning the *Acceleration mission* track layout in Paragraph 2.1, it has set the *If* cycle considering a reference *rect_area* value of 4.6 $m^2$. Analyzing each frame at a time, if the calculated rectangular area resulting from it, is less than 4.6 $m^2$, the frame perceived by the Lidar sensor is a straight and therefore the *rd_v_potGoal* value is set to 1.2 m, otherwise if the area is greater than 4.6 $m^2$, the frame defines a probable curve and therefore this distance *rd_v_potGoal* is set at 1.4 m. *PotPoints* that have a distance along the y-axis from the vehicle position ($origin = [0, 0]$) that is less than the *rd_v_potGoal* value defined with the method above are collected in an array called *PotGoals*.

In some rare cases, it may happen that the *PotGoals* array is empty. This happens because each *potential point* has a relative distance in the y direction from the racecar position greater than the limit defined by *rd_v_potGoal* value. In this case the strategy implemented is different. Firstly it is calculated the maximum and minimum abscissa

from the Cartesian coordinates of the obstacles that define the specific frame. Subsequently, the average abscissa is calculated starting from the two previously calculated values and on the basis of this midpoint ($h\_Dx$), the *PotPoints* which have their abscissa far from $h\_Dx$ within 1 m are searched. Thus, at least one *potential goal* is defined without the algorithm going into error. The Figure 3.14 shows the particular case in question. In Figure 3.15 are represented the potential goals calculated with

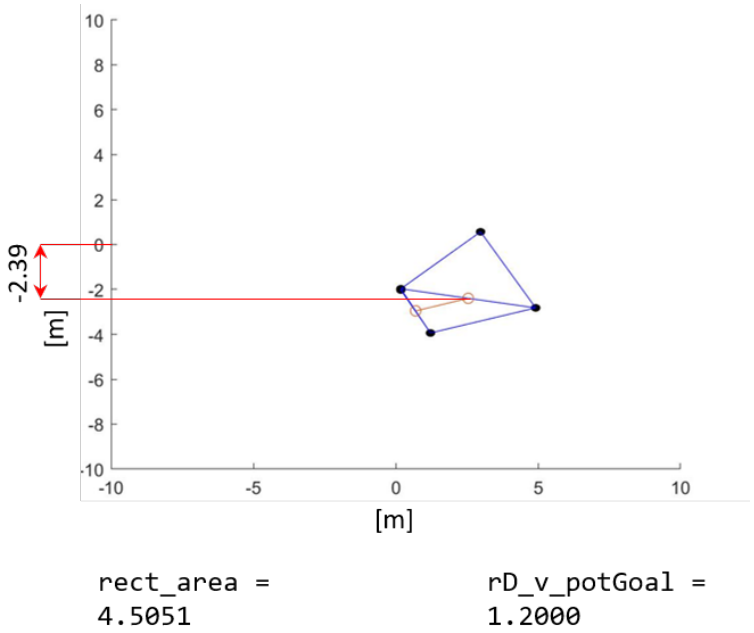



**Figure 3.14:** *Particular condition (Frame n° 509) in which PotPoints have a relative distance in y direction respect to the vehicle position higher than the rd_v_potGoal value.*

the method explained above. Algorithm 3 depicts the *PotGoals* procedure.

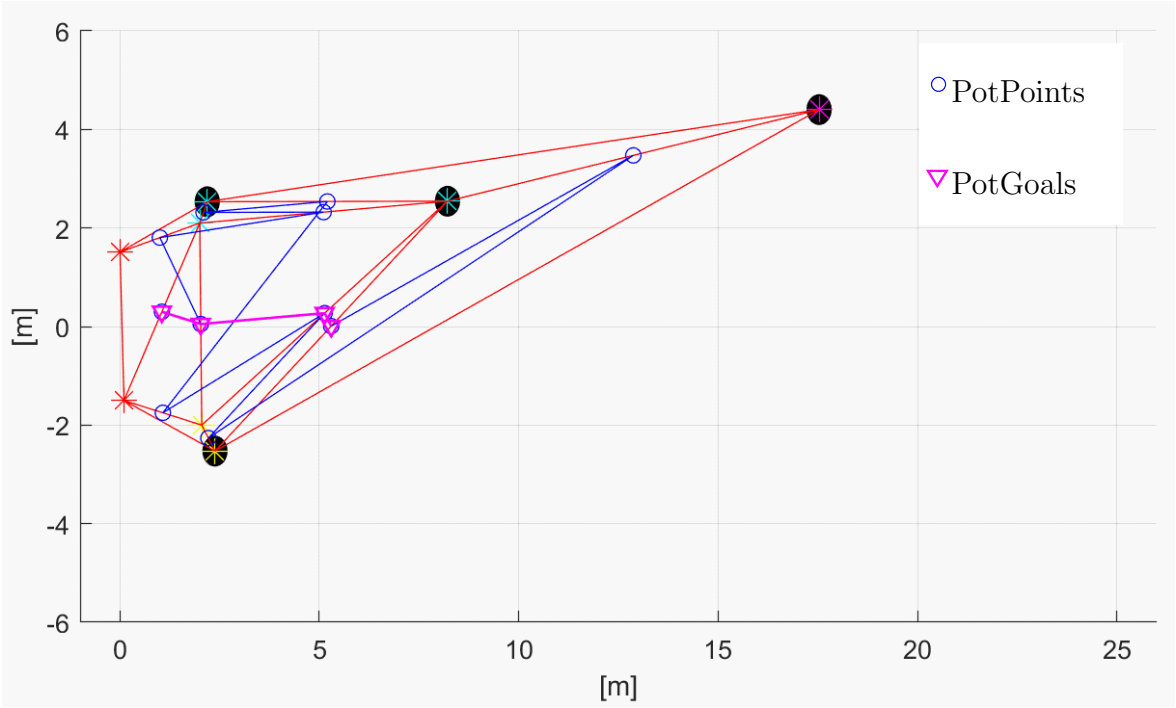



**Figure 3.15:** *Calculation of Potential Goals (PotGoals).*

**Algorithm 3** Potential Goals calculation

```
1:  procedure POTGOALS(CList, PotPoints, origin, rD_v_potGoal, obs)
2:      for i ← 1, size(PotPoints, 1) do
3:          if abs(origin(2) − PotPoints(i, 2)) < rD_v_potGoal then
4:              PotGoals = PotPoints(i, :);
5:          end if
6:      end for
7:      if size(PotGoals) == 0 then
8:          x_max = max(obs(:, 1));
9:          x_min = min(obs(:, 1));
10:         h_Dx = (x_max − x_min)/2;
11:         for i ← 1, size(PotPoints, 1) do
12:             if abs(h_Dx − PotPoints(i, 1)) < 1 then
13:                 PotGoals = PotPoints(i, :);
14:             end if
15:         end for
16:     end if
17: end procedure
```

### Final Goal - *finalGoal*

After calculating the *Potgoals* array, we are able to extrapolate from these what will define the final configuration for the calculation of the trajectory for the specific frame. Since the mission in question is the *Acceleration* one, it is necessary to define a goal that is at a distance ahead of the vehicle such that the updated position of the vehicle never exceeds that one of the goal. The *Path Planner* block updates the goal with a frequency of 10 Hz, the frequency with which the LiDAR sensor sends the Cartesian coordinates of the perceived obstacles to the block. For example, reaching a target speed of 70 km/h in this *mission*, means the vehicle, every 0.1 s, will travel about 2 m. To be conservative and to avoid the condition in which the goal calculated is already overtaken by the racecar, a code that allows to define a goal always 20 m forward of the vehicle position has been implemented.
As first step, from the *PotGoals* array it is extrapolated the *potential goal* at further euclidean distance from the position of the racecar, which for each frame corresponds to the $origin = [0, 0]$. Then the *finalGoal* will be this *potential goal* setting the abscissa coordinate at 20 m ahaead respect to the x-position of the racecar. The Algorithm 4 represents the procedure for calculating the final vehicle configuration for each frame. In Figure 3.16 the last step for the automatic goal generation is shown.

**Algorithm 4** Final Goal calculation

1: **procedure** FINALGOAL($PotGoals, origin$)
2: **for** $i \leftarrow 1, size(PotGoals, 1)$ **do**
3: $d\_origin\_potgoals = ((origin(1,1) - PotGoals(i,1))^2 + (origin(1,2) - PotGoals(i,2))^2)$;
4: **end for**
5: $[maxdistance,\ IDPotGoal] = max(d\_origin\_potgoals)$
6: $Goal = PotGoals(IDpotGoal, :)$;
7: $finalGoal = [(Goal(:,1) + (20 - Goal(:,1))), Goal(:,2)]$
8: **end procedure**

Whenever the circuit layout perceived by the LiDAR sensor is a straight path, the Algorithm defined as written above generates always a *finalGoal* in the middle of the track width, 20 m far in front of the vehicle position. Analying more than 3000 frames, it can be said that the *finalgoal* is never positioned on the side of the circuit and is always consistent with the future generation of the trajectory the vehicle will follow.

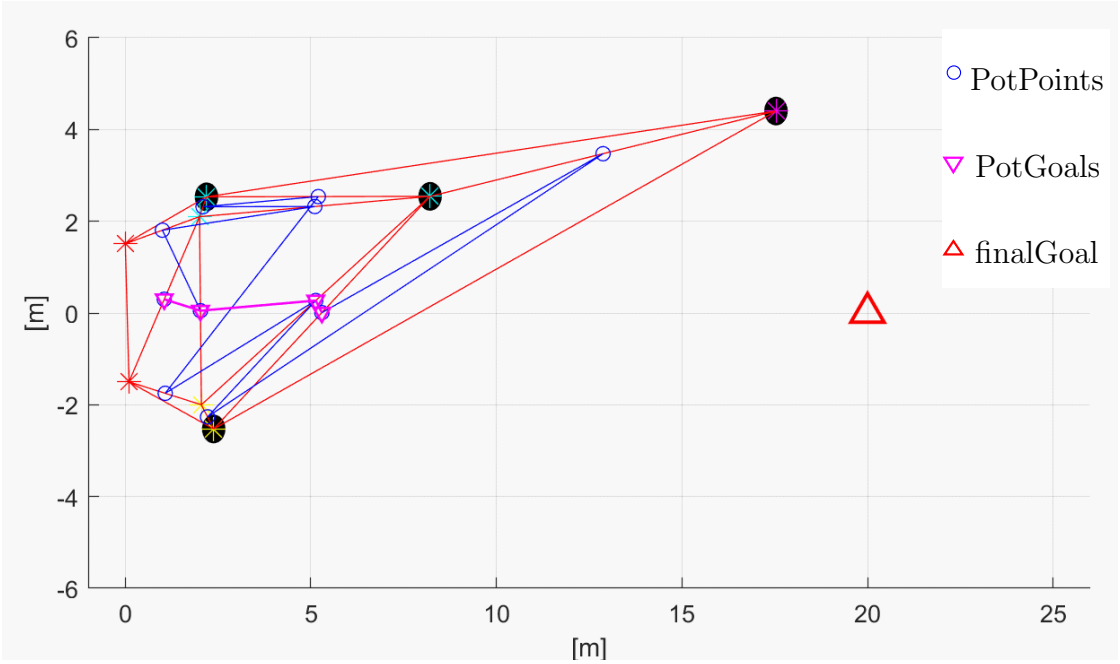
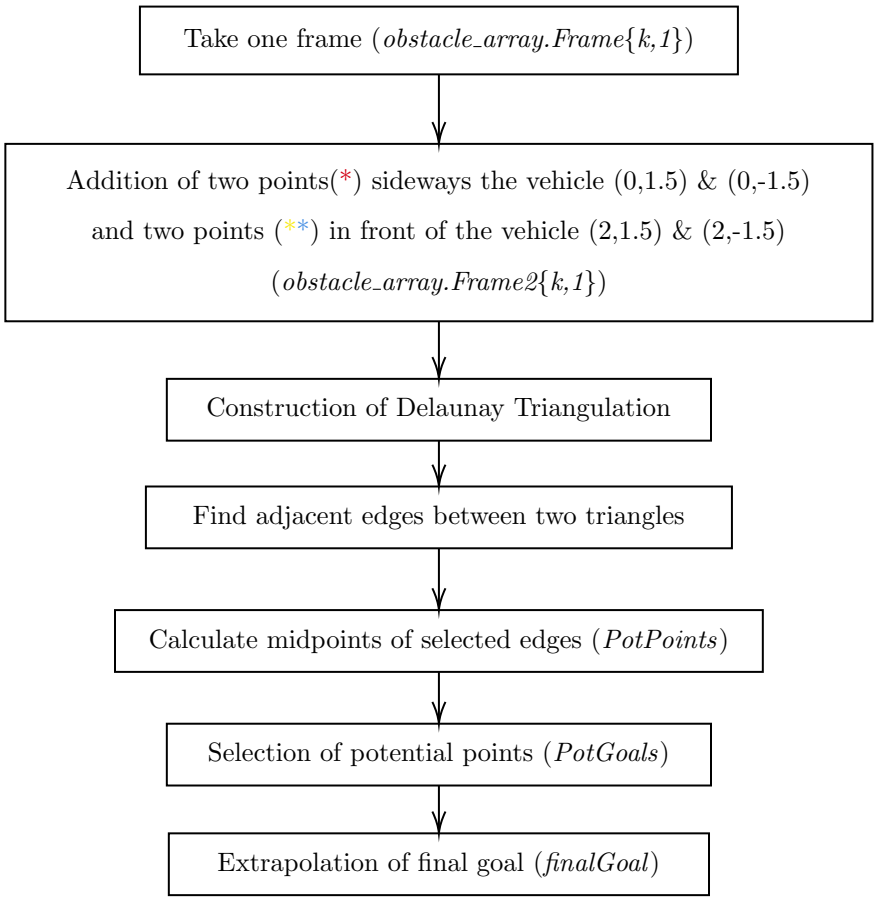



**Figure 3.16:** *Calculation of the final configuration for racecar path computation (finalGoal).*

Below it is reported a summary diagram of the procedure for the automatic goal generation.



### 3.2.2 Generation of the modified RRT algorithm with Dubins curves

Most of the wheeled mobile robots don't have differential drive system, which means they can't rotate in one place. Car-like robots have wheels that are required to roll in the direction they are pointing, and they are not designed to slide sideways. Therefore, this implies velocity constraints on rolling vehicles, called differential constraints, and those mobile robots usually have less action variables then degrees of freedom. This kind of robots is called non-holonomic, or *underactuated*. The model of a *Simple car* is one of the examples of non-holonomic mobile robot with differential constraints [30], Figure 3.17. The car can be imagined as a rigid body that moves in the plane.

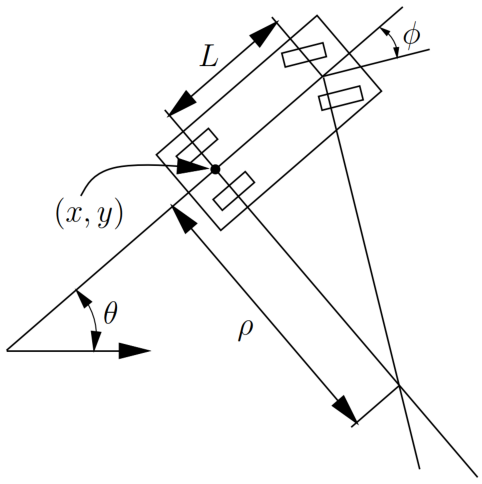



**Figure 3.17:** *Simple car model.*

Therefore, its $\mathcal{C}$-space is $\mathcal{C} = \mathbb{C}^2 \times \mathbb{S}^1$. A configuration is denoted by $q = (x,\ y,\ \theta)$. The body frame of the car places the origin at the center of rear axle, and the x-axis points along the main axis of the car. Let $s$ denote the speed of the car and $\phi$ the steering angle. The distance between the front and rear axles is represented as L. If the steering angle is fixed at $\phi$, the car travels in a circular motion, in which the radius of the circle is $\rho$.

### Dubins Car

If the speed of the *Simple car* model is restricted to have only positive values, which means that the robot can move only forward, the model of *Dubins car* is obtained [42]. Using the notation in Figure 3.17, the vehicle motion can be described using time derivative of configuration vector q as a set of equations in the following form [30]:

$$\begin{aligned} \dot{x} &= f_1(x, y, \theta, s, \phi) \\ \dot{y} &= f_2(x, y, \theta, s, \phi) \\ \dot{\theta} &= f_2(x, y, \theta, s, \phi) \end{aligned} \tag{3.1}$$

Exploiting the speed of the car $s$ and geometry from Figure 3.17, equations for $x$ and $y$ are obtaind as $\dot{x} = s\cos\theta$ and $\dot{y} = s\sin\theta$. Using the steering angle $\phi$, speed of the vehicle $s$ and distance between front and rear axles $L$, equation for angular velocity of the $\theta$ is produced as $\dot{\theta} = \frac{s}{L}\tan\phi$.
Since the speed $s$ and the steering angle $\phi$ are only two variables that can be controlled, the control variables are defined using two dimensional vector as follows:

$$u = (u_s, u_\phi) \tag{3.2}$$

Using notation given in (3.2), the equation of the robot motion in configuration space is obtained as follows:

$$\begin{aligned} \dot{x} &= u_s\cos\theta \\ \dot{y} &= u_s\sin\theta \\ \dot{\theta} &= \frac{u_s}{L}\tan u_\phi \end{aligned} \tag{3.3}$$

where (3.3) describes a mathematical model of the *Simple car*'s motion. In order to complete this model it is needed to specify allowed ranges for control variables. Considering steering angle $\phi$, under assumption that the vehicle can't rotate in one place, it is obvious that maximum steering angle should satisfy the following constraint:

$$\phi_{max} < \frac{\pi}{2} \tag{3.4}$$

Therefore it is required that the steering angle takes value according to:

$$|\phi| \leq \phi_{max} \tag{3.5}$$

Further, as already said, there is the assumption that the vehicle can move only forward, which is constraint made by *Dubins car* model. Therefore, the speed should take any positive value, $u_s \in \mathbb{R}$. To simplify this case for the purpose of the algorithm, velocity will be considered as constant with possible values as following:

$$u_s \in \{0, 1\} \tag{3.6}$$

when $u_s$ has zero value, it means the car is not moving, while value one denotes moving forward with a constant speed. The equation given in (3.3) along with constraints in (3.4), (3.5) and (3.6) represents the mathematical model of *Dubins car*. Whole procedure obtaining this model is given in [30].

### Dubins curves

Due to differential constraints presented before, the basic form of RRT algorithm for path planning is unusable. The main reason is because *Dubins car* is unable to rotate in one place, so it's not possible to reach all configurations from specific state. One of the solutions to modify algorithm to meet those constraints is using Dubins curve when building search tree. It means that every branch in this case represents Dubins curve instead of straight line. Considering the model of *Dubins car*, the vehicle moves with a constant forward speed, $u_s = 1$, and has maximum steering angle $\phi_{max}$, which results in a minimum turning radius $\rho_{min}$. As the car travels, the path descibed by the vehicle is considered tracking a reference point located on the center of the rear axle. Assuming that the car travels from initial $\mathrm{q}_I$ to final point $\mathrm{q}_F$, the main task is to minimize the length of the curve between those two points. In order to have circular trajectory, considering that the $\rho_{min} < 0$, the following cost function is exploited:

$$L\left(\tilde{q}, \tilde{u}\right) = \int_0^{t_F} \sqrt{\dot{x}\left(t\right)^2 + \dot{y}\left(t\right)^2}\, dt \tag{3.7}$$

**Table 3.3:** *The three motion primitives from which all optimal curves for the Dubins car can be constructed.*

| **Symbol** | **Steering**: u |
|---|---|
| S | 0 |
| L | 1 |
| R | -1 |

in which $t_F$ is the time in which $q_G$ is reached. If $q_G$ is not reached, then it is assumed that $L(\tilde{q}, \tilde{u}) = \infty$. Since the speed is constant the system can be semplified to

$$\begin{aligned} \dot{x} &= \cos\theta \\ \dot{y} &= \sin\theta \\ \dot{\theta} &= u \end{aligned} \tag{3.8}$$

in which $u$ is chosen from the interval $U = [-\tan\phi_{max}, \tan\phi_{max}]$. This implies that (3.7) reduces to optimizing the time $t_F$ to reach $q_G$ because the integrand reduces to 1. For simplicity, assume that $\tan\phi = 1$.

It was shown in [12] that between any two configurations, the shortest path for the *Dubins Car* can always be expressed as a combination of no more than three primitives. Each motion primitive applies a constant action over an interval of time. Furthermore, the only actions that are needed to traverse the shortest paths are $u \in \{-1, 0, 1\}$. The primitives and their associated symbols are shown in Table 3.3. The S primitive drives the car straight ahead (straight line of motion S). The L and R primitives turn as sharply as possible to the left and right, respectively (both along a circle C of radius $\rho_{min}$). Using these symbols, each possible kind of shortest path can be designated as a sequence of three symbols that corresponds to the order in which the primitives are applied. Let such a sequence be called a *word* [30].There is no need to have two consecutive primitives of the same kind because they can be merged into one. Under this observation, ten possible words of length three are possible. Dubins showed that only these six words are possibly optimal:

$$\{LRL, RLR, LSL, LSR, RSL, RSR\} \tag{3.9}$$

The shortest path between any two configurations can always be characterized by one of these words. These are called the *Dubins curves*, Figure 3.18. Furthermore, Dubins'

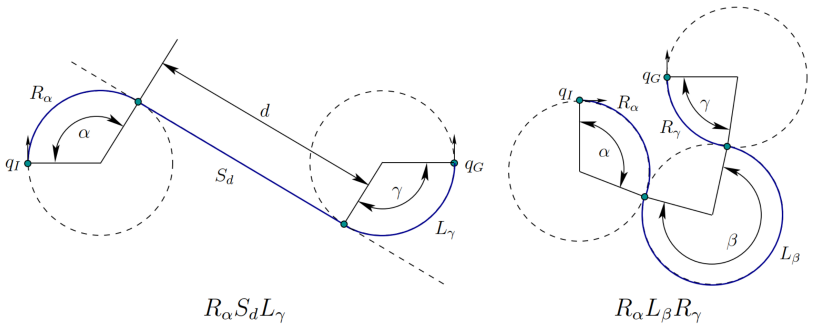



**Figure 3.18:** *The trajectories for two words are shown in* $\mathcal{W} = \mathbb{R}^2$.

theorem states that in order to be a candidate for the optimal path, each arc must be of the minimal allowed radius $\rho$.

To use Dubins's result for the shortest path calculation, one would need to explicitly calculate the lengths of all arcs and straight line segments in the Dubins set, and then choose the shortest of the computed paths. The time necessary for this calculation may become a bottleneck in time-constrained applications, as e.g. in real-time robot motion planning, but fortunately it is not an issue for our application.

So, the three corresponding operators $L_\upsilon$ (for left turn), $R_\upsilon$ (for right turn), $S_\upsilon$ (for straight), transform an arbirary point $(x, y, \phi) \in \mathbb{R}^3$ into its corresponding image point in $\mathbb{R}^3$,

$$\begin{aligned} L_\upsilon(x, y, \phi) &= (x + \sin(\phi + \upsilon) - \sin\phi, y - \cos(\phi + \upsilon) + \cos\phi, \phi + \upsilon) \\ R_\upsilon(x, y, \phi) &= (x - \sin(\phi - \upsilon) + \sin\phi, y + \cos(\phi - \upsilon) - \cos\phi, \phi - \upsilon) \\ S_\upsilon(x, y, \phi) &= (x + \upsilon\cos\phi, y + \upsilon\sin\phi, \phi) \end{aligned} \tag{3.10}$$

where index $\upsilon$ indicates that the motion has been along the (C or S) segment of lenght $\upsilon$. With these elementary transformations, any path in in the Dubins set $\mathcal{D} = \{LRL, RLR, LSL, LSR, RSL, RSR\}$ can be expressed in terms of the corresponding equations. In the coordinate system chosen, the initial configuration of each path is at $(0, 0, \alpha)$ and the final configuration at $(d, 0, \beta)$, as reported in Figure 3.19. For example, a path made of segments L, R and L, of the lenghts t, p, q, respectively, which starts at

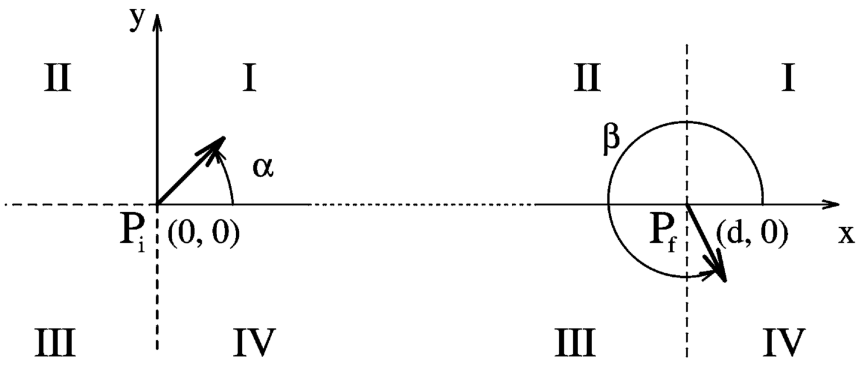



**Figure 3.19:** *The coordinate system, the initial configuration $(P_i, \alpha)$ and the final configuration $(P_f, \beta)$. Possible orientation angles are divided into four quadrants.*

point $(0, 0, \alpha)$, must end at $L_{\mathrm{q}}\left(R_{\mathrm{p}}\left(L_{\mathrm{t}}\left(0, 0, \alpha\right)\right)\right) = (d, 0, \beta)$. The lenght $\mathcal{L}$ of the path can be defined as the sum of lenght t, p and q of its constituent segments,

$$\mathcal{L} = \mathrm{t} + \mathrm{p} + \mathrm{q} \tag{3.11}$$

An explicit computation of all candidates for the shortest path will follow below. To this end, elements of $\mathcal{D}$ will be considered one-by-one and the operator equations for the lenghts of each path derived.

1. $L_{\mathrm{q}}\left(S_{\mathrm{p}}\left(L_{\mathrm{t}}\left(0, 0, \alpha\right)\right)\right) = (d, 0, \beta)$. By appling the corresponding operators (3.10), this first path in $\mathcal{D}$ can be represented by a system of three scalar equations:

$$\begin{aligned} p\cos(\alpha + \mathrm{t}) - \sin\alpha + \sin\beta &= \mathrm{d} \\ p\sin(\alpha + \mathrm{t}) + \cos\alpha - \cos\beta &= 0 \\ \alpha + \mathrm{t} + \mathrm{q} &= \beta \,\{mod 2\pi\} \end{aligned} \tag{3.12}$$

   The solution of this system with respect to the segments t, p and q is found as

$$\begin{aligned} \mathrm{t}_{lsl} &= -\alpha + \arctan\frac{\cos\beta - \cos\alpha}{\mathrm{d} + \sin\alpha - \sin\beta}\,\{mod 2\pi\} \\ \mathrm{p}_{lsl} &= \sqrt{2 + \mathrm{d}^2 - 2\cos(\alpha - \beta) + 2\mathrm{d}(\sin\alpha - \sin\beta)} \\ \mathrm{q}_{lsl} &= \beta - \arctan\frac{\cos\beta - \cos\alpha}{\mathrm{d} + \sin\alpha - \sin\beta}\,\{mod 2\pi\} \end{aligned} \tag{3.13}$$

   Using definition (3.11), the lenght of the path LSL as a function of the boundary

conditions can be now written as

$$\mathcal{L}_{lsl} = \mathrm{t}_{lsl} + \mathrm{p}_{lsl} + \mathrm{q}_{lsl} = -\alpha + \beta + \mathrm{p}_{lsl} \tag{3.14}$$

2. $R_{\mathrm{q}}\left(S_{\mathrm{p}}\left(R_{\mathrm{t}}\left(0,0,\alpha\right)\right)\right) = (d,0,\beta)$. Using 3.10, we obtain the corresponding scalar equations:

$$\begin{gathered} p\cos(\alpha-\mathrm{t}) + \sin\alpha - \sin\beta = \mathrm{d} \\ p\sin(\alpha-\mathrm{t}) - \cos\alpha + \cos\beta = 0 \\ \alpha - \mathrm{t} - \mathrm{q} = \beta\,\{mod2\pi\} \end{gathered} \tag{3.15}$$

The solution of this system, i.e. the lenghts of the corresponding segments, is

$$\begin{gathered} \mathrm{t}_{rsr} = \alpha - \arctan\frac{\cos\alpha - \cos\beta}{\mathrm{d} - \sin\alpha + \sin\beta}\,\{mod2\pi\} \\ \mathrm{p}_{rsr} = \sqrt{2 + \mathrm{d}^2 - 2\cos(\alpha-\beta) + 2\mathrm{d}(\sin\beta - \sin\alpha)} \\ \mathrm{q}_{rsr} = -\beta(mod2\pi) + \arctan\frac{\cos\alpha - \cos\beta}{\mathrm{d} - \sin\alpha + \sin\beta}\,\{mod2\pi\} \end{gathered} \tag{3.16}$$

and the path lenght is given by

$$\mathcal{L}_{rsr} = \mathrm{t}_{rsr} + \mathrm{p}_{rsr} + \mathrm{q}_{rsr} = \alpha - \beta + \mathrm{p}_{rsr} \tag{3.17}$$

3. $R_{\mathrm{q}}\left(S_{\mathrm{p}}\left(L_{\mathrm{t}}\left(0,0,\alpha\right)\right)\right) = (d,0,\beta)$. Using (3.10), we obtain the corresponding scalar equations:

$$\begin{gathered} p\cos(\alpha+\mathrm{t}) + 2\sin(\alpha+\mathrm{t}) - \sin\alpha - \sin\beta = \mathrm{d} \\ p\sin(\alpha+\mathrm{t}) - 2\cos(\alpha+\mathrm{t}) + \cos\alpha + \cos\beta = 0 \\ \alpha + \mathrm{t} - \mathrm{q} = \beta\,\{mod2\pi\} \end{gathered} \tag{3.18}$$

The solution of this system is

$$\begin{gathered} \mathrm{t}_{lsr} = \left(-\alpha + \arctan\left(\frac{-\cos\alpha - \cos\beta}{\mathrm{d} + \sin\alpha + \sin\beta}\right) - \arctan\left(\frac{-2}{\mathrm{p}_{lsr}}\right)\right)\{mod2\pi\} \\ \mathrm{p}_{lsr} = \sqrt{-2 + \mathrm{d}^2 + 2\cos(\alpha-\beta) + 2\mathrm{d}(\sin\alpha + \sin\beta)} \\ \mathrm{q}_{lsr} = -\beta(mod2\pi) + \arctan\left(\frac{-\cos\alpha - \cos\beta}{\mathrm{d} + \sin\alpha + \sin\beta}\right) - \arctan\left(\frac{-2}{\mathrm{p}_{lsr}}\right)\{mod2\pi\} \end{gathered} \tag{3.19}$$

and the path lenght is given by

$$\mathcal{L}_{lsr} = \mathrm{t}_{lsr} + \mathrm{p}_{lsr} + \mathrm{q}_{lsr} = \alpha - \beta + 2\mathrm{t}_{lsr} + \mathrm{p}_{lsr} \tag{3.20}$$

4. $L_{\mathrm{q}}\left(S_{\mathrm{p}}\left(R_{\mathrm{t}}\left(0,0,\alpha\right)\right)\right) = (d,0,\beta)$. Using (3.10) we obtain the corresponding scalar equations:

$$\begin{gathered} p\cos(\alpha - \mathrm{t}) - 2\sin(\alpha - \mathrm{t}) + \sin\alpha + \sin\beta = \mathrm{d} \\ p\sin(\alpha - \mathrm{t}) + 2\cos(\alpha - \mathrm{t}) - \cos\alpha - \cos\beta = 0 \\ \alpha - \mathrm{t} + \mathrm{q} = \beta\,\{mod2\pi\} \end{gathered} \tag{3.21}$$

The corresponding solution is

$$\begin{gathered} \mathrm{t}_{rsl} = \alpha - \arctan\left(\frac{\cos\alpha + \cos\beta}{\mathrm{d} - \sin\alpha - \sin\beta}\right) + \arctan\left(\frac{2}{\mathrm{p}_{rsl}}\right)\{mod2\pi\} \\ \mathrm{p}_{rsl} = \sqrt{\mathrm{d}^2 - 2 + 2\cos(\alpha - \beta) - 2\mathrm{d}(\sin\alpha + \sin\beta)} \\ \mathrm{q}_{rsl} = \beta(mod2\pi) - \arctan\left(\frac{\cos\alpha + \cos\beta}{\mathrm{d} - \sin\alpha - \sin\beta}\right) - \arctan\left(\frac{2}{\mathrm{p}_{rsl}}\right)\{mod2\pi\} \end{gathered} \tag{3.22}$$

and the path lenght is given by

$$\mathcal{L}_{rsl} = \mathrm{t}_{rsl} + \mathrm{p}_{rsl} + \mathrm{q}_{rsl} = -\alpha + \beta + 2\mathrm{t}_{rsl} + \mathrm{p}_{rsl} \tag{3.23}$$

5. $R_{\mathrm{q}}\left(L_{\mathrm{p}}\left(R_{\mathrm{t}}\left(0,0,\alpha\right)\right)\right) = (d,0,\beta)$. Using (3.10) we obtain the corresponding scalar equations:

$$\begin{gathered} 2\sin\left(\alpha - \mathrm{t} + \mathrm{p}\right) - 2\sin\left(\alpha - \mathrm{t}\right) = \mathrm{d} - \sin\alpha + \sin\beta \\ -2\cos\left(\alpha - \mathrm{t} + \mathrm{p}\right) + 2\cos\left(\alpha - \mathrm{t}\right) = \cos\alpha - \cos\beta \\ \alpha - \mathrm{t} + \mathrm{p} - \mathrm{q} = \beta\,\{mod2\pi\} \end{gathered} \tag{3.24}$$

The corresponding solution is

$$\begin{gathered} \mathrm{t}_{rlr} = \alpha - \arctan\left(\frac{\cos\alpha - \cos\beta}{\mathrm{d} - \sin\alpha + \sin\beta}\right) + \frac{\mathrm{p}_{rlr}}{2}\,\{mod2\pi\} \\ \mathrm{p}_{rlr} = \arccos\frac{1}{8}\left(6 - d^2 + 2\cos\left(\alpha - \beta\right) + 2d\left(\sin\alpha - \sin\beta\right)\right) \\ \mathrm{q}_{rlr} = \alpha - \beta - \mathrm{t}_{rlr} + \mathrm{p}_{rlr}\,\{mod2\pi\} \end{gathered} \tag{3.25}$$

and the path lenght is obtained by substituting (3.25) into (3.11)

$$\mathcal{L}_{rlr} = \mathrm{t}_{rlr} + \mathrm{p}_{rlr} + \mathrm{q}_{rlr} = \alpha - \beta + 2\mathrm{p}_{rlr} \tag{3.26}$$

6. $L_\mathrm{q}\left(R_\mathrm{p}\left(L_\mathrm{t}\left(0,0,\alpha\right)\right)\right) = (d,0,\beta)$. Using (3.10) we obtain the corresponding scalar equations:

$$\begin{gathered} -2\sin\left(\alpha+\mathrm{t}-\mathrm{p}\right)+2\sin\left(\alpha+\mathrm{t}\right)=\mathrm{d}+\sin\alpha-\sin\beta \\ 2\cos\left(\alpha+\mathrm{t}-\mathrm{p}\right)-2\cos\left(\alpha+\mathrm{t}\right)=-\cos\alpha+\cos\beta \\ \alpha+\mathrm{t}-\mathrm{p}+\mathrm{q}=\beta\left\{mod2\pi\right\} \end{gathered} \tag{3.27}$$

The corresponding solution is

$$\begin{gathered} \mathrm{t}_{lrl}=\left(-\alpha+\arctan\left(\frac{-\cos\alpha+\cos\alpha}{\mathrm{d}+\sin\alpha-\sin\beta}\right)+\frac{\mathrm{p}_{lrl}}{2}\right)\left\{mod2\pi\right\} \\ \mathrm{p}_{lrl}=\arccos\frac{1}{8}\left(6-d^2+2\cos\left(\alpha-\beta\right)+2d\left(\sin\alpha-\sin\beta\right)\right)\left\{mod2\pi\right\} \\ \mathrm{q}_{lrl}=\beta\left(mod2\pi\right)-\alpha+2\mathrm{p}_{lrl}\left\{mod2\pi\right\} \end{gathered} \tag{3.28}$$

and the path lenght is given by

$$\mathcal{L}_{lrl} = \mathrm{t}_{lrl} + \mathrm{p}_{lrl} + \mathrm{q}_{lrl} = -\alpha + \beta + 2\mathrm{p}_{lrl} \tag{3.29}$$

### Dubins curves generation Procedure

Before starting with the description of the various steps that lead to the creation of the Dubins curves, the parameters suitable for the initialization of the algorithm are defined. Every 0.1 s, the *Local Path planner* calculates the *Dubins curves* with initial configuration $\mathrm{q}_I$ the $origin = (0,0,0)$ and final configuration $\mathrm{q}_G$, the *finalGoal* defined in 3.2.1. Considering the layout of the *Acceleration mission* in question, the *Map* within which the random points are calculated has the following values, Table 3.4: A map with a width of 6 m and a length of 25 m was chosen in order to obtain paths that have lengths of at least 25 m with minimal oscillations along the y direction, since theoretically the vehicle has to cover only a straight in the shortest possible

**Table 3.4:** *Map sizes paramenters in order to define the space within which the random vertices of the Dubins curves are calculated.*

| Map Size | Value |
|---|---|
| map.height | 6 |
| map.width | 25 |
| map.center | [map.width/2,0] |
| map.offset | [0,-3] |

**Table 3.5:** *Dubins curves parameters for paths genertion.*

| Dubins parameters | Value |
|---|---|
| iteration | 200 |
| th_range | 10*pi/180 |
| th_center | 0 |
| th_offset | -5*pi/180 |
| turning_rad | pi |
| expansion | 0 |

time. Moreover, the length of the future paths created in the x direction helps to avoid entering the error condition in which the vehicle has already passed the *finalGoal* position.

Given the speeds involved, a number of iterations has been defined for the calculation of the random vertices of the Dubins curves as a trade-off between the computation speed of the path in a very short period of time (0.1 s) and the effectiveness with which we generate a path consistent with the layout of the specific *mission*. Other parameters, Table 3.5, in order to generate the *Dubins curves*, were decided on the basis of the vehicle's steering specifications such as the minimum turning radius. First of all, the random points lying inside the map are calculated, whose parameters have been defined in the Table 3.4. Subsequently, it is verified that these points are located at a distance with respect to the Cartesian coordinates of the obstacles perceived by the LiDAR sensor greater than a safety value, called in this case *inflation_rad*, Algorithm 5. This will allow to obtain paths that will have vertexes that certainly do not collide with obstacles. After verifying that the Cartesian coordinates of the random point do not collide with the obstacles, the calculation of the *Dubins curve* begins, Algorithm 6. The curve has as extremes the $origin = [0, 0, 0]$ and the random point calculated above and radius of curvature for the circle paths $\rho$ equal to the *turning_rad* value defined in the Table 3.5. In Algorithm 6, the parameters t, p and q for each curve

**Algorithm 5** Check available x_y random point

```
1: procedure CHECK_AVAILABLE_XY(point_rand, origin, obs, inflation_rad)
2:     for i ← 1, size(obs, 1) do
3:         for j ← 1, size(point_rand, 1) do
4:             if (sqrt(((point_rand(j,1) − obs(i,1))^2) + ((point_rand(j,2) − obs(i,2))^2))) <= inflation_rad then
5:                 check_available_xy = 0;
6: return
7:             end if
8:         end for
9:     end for
10:    check_available_xy = 1;
11: end procedure
```

belonging to the Dubins set $\mathcal{D}$ are calculated according to the formulation previously defined in Paragraph 3.2.2. The sum of these three parameters for each of the 6 *words* is calculated. The *word* with the lowest sum of the three parameters is chosen as the best curve and therefore the resulting parameters saved. The length of the curve is then subsequently calculated, Algorithm 7. Remember that *param* is a structure with 5 fields:

1. *p_init*: initial 3D pose, starting from point $origin = [0, 0, 0]$;
2. *seg_param*: t, p and q paramenters of the Dubins curve that has the best cost between the 6 *words* and extremes the *origin* and the n$^{\text{th}}$ random point;
3. $r$: radius $\rho$ of the circle section of the *word* that composed the Dubins curve;
4. *type*: six types of curves, $LSL = 1$, $LSR = 2$, $RSL = 3$ $RSR = 4$, $RLR = 5$, $LRL = 6$;
5. *flag*: $flag = 0$ means to continue the calculations, $flag = 1$ means to stop.

Once the Dubins curve has been calculated, it must be checked that it does not collide with the obstacles sensed by the Lidar Sensor, Algorithm 8. An array called *ind_nearest_tree* is created in which the newly generated point is appended; it will contains the n$^{\text{th}}$ nearest vertex respect to the (n+1)$^{\text{th}}$ one. Then it is also created an array called *edges.param* p-by-8 where the columns define:

**Algorithm 6** Dubins curve calculation

**procedure** PARAM($vertecies(1,:), rand, rad$)
    $param.p\_init = p1$;
    $param.seg_param = [0,0,0]$;
    $param.r = r$;
    $param.type = -1$;
    $param.flag = 0$;
    $dx = p2(1) - p1(1)$;
    $dy = p2(2) - p1(2)$;
    $D = sqrt(dx^2 + dy^2)$;
    $d = D/r$;
    $theta = mod(atan2(dy, dx), 2 * \pi)$;
    $alpha = mod((p1(3) - theta), 2 * \pi)$;
    $beta = mod((p2(3) - theta), 2 * \pi)$;
    $test\_param(1,:) = dubins\_LSL(alpha, beta, d)$;
    $test\_param(2,:) = dubins\_LSR(alpha, beta, d)$;
    $test\_param(3,:) = dubins\_RSL(alpha, beta, d)$;
    $test\_param(4,:) = dubins\_RSR(alpha, beta, d)$;
    $test\_param(5,:) = dubins\_RLR(alpha, beta, d)$;
    $test\_param(6,:) = dubins\_LRL(alpha, beta, d)$;
    **for** $i \leftarrow 1, 6$ **do**
        **if** $(test\_param(i,1)) \sim= -1$ **then**
            $cost = sum(test\_param(i,:))$;
            **if then**$(cost < best\_cost) || (best\_cost == -1)$
                $best\_word = i$;
                $best\_cost = cost$;
                $param.seg\_param = test\_param(i,:)$;
                $param.type = i$
            **end if**
        **end if**
    **end for**
    **if then**$best\_word == -1$
        $param.flag = -2$  $\%NO\ PATH$
**return** ;
    **else**
    **end if**
**end procedure**

**Algorithm 7** Dubins curve lenght calculation

**procedure** LENGHT($param$)
    $length = param.seg\_param(1) + param.seg\_param(2) + param.seg\_param(3)$;
    $length = lenght * param.r$;
**end procedure**

**Algorithm 8** Check collision between Dubins curve and obstacles

```
1: procedure COLLIDE(param, map, obs, exp, inflation_rad)
2:     for seg_i ← 1, 3 do
3:         seg_type = DIRDATA(param.type, seg_i);
4:         this_end = dubins_get_xyt(param.seg_param(seg_i), last_end, seg_type, param.r);
5:         if (chk_map_range(this_end, map, exp) == 0) then
6:             collide = 1;
7: return
8:         end if
9:         if (seg_type == L_SEG) then
10:            arc.x = last_end(1) − param.r ∗ sin(last_end(3));
11:            arc.y = last_end(2) + param.r ∗ cos(last_end(3));
12:            arc.ang_init = last_end(3) − pi/2;
13:            arc.ang_end = last_end(3) + param.seg_param(seg_i) − pi/2;
14:            collide = chk_arc_poly_collision(arc, obs, exp, inflation_rad);
15:        else if seg_type == R_SEG then
16:            same procedure for seg_type == L_SEG
17:        else if seg_type == S_SEG then
18:            collide = chk_line_exp_collision([last_end(1 : 2); this_end(1 : 2)], obs, exp, inflation_rad);
19:        else
20:            error("error segment type");
21: return
22:        end if
23:        if (collide) then
24: return
25:        end if
26:        last_end = this_end;
27:    end for
28:    collide = 0;
29: return
30: end procedure
```

**Table 3.6:** *Table representing the Dubins paramenters after 200 iterations. The Dubins curve tree expands starting from the origin, initial configuration* $q_I$*. The few vertices found are due to a relatively small map.*

| **param** | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| x | y | theta | t | p | q | rho | type | flag |
| 0 | 0 | 6.27 | 0.087 | 1.362 | 0.059 | 3.141 | 2 | 0 |
| 4.730 | 0.334 | 0.015 | 0.066 | 3.732 | 0.072 | 3.141 | 3 | 0 |
| 16.877 | -0.277 | 0.020 | 0.263 | 0.673 | 0.170 | 3.141 | 3 | 0 |
| 16.877 | -0.277 | 0.020 | 0.060 | 2.203 | 0.128 | 3.141 | 2 | 0 |
| 4.730 | 0.334 | 0.015 | 0.242 | 1.313 | 0.187 | 3.141 | 3 | 0 |
| 0 | 0 | 6.27 | 0.039 | 2.007 | 0.008 | 3.141 | 4 | 0 |
| 4.730 | 0.334 | 0.015 | 0.464 | 0.346 | 0.516 | 3.141 | 3 | 0 |
| 0 | 0 | 6.27 | 0.213 | 0.442 | 6.015 | 3.141 | 4 | 0 |
| 16.877 | -0.277 | 0.020 | 0.113 | 0.743 | 0.102 | 3.141 | 2 | 0 |
| 20.278 | -0.961 | -0.072 | 0.124 | 0.446 | 0.086 | 3.141 | 2 | 0 |

1. *p_init*: $x$, $y$, $\theta$ of the random point which has passed all the checks described above;
2. *seg_param_temp*: t, p and q parameters of the *Dubins curve*;
3. *r_temp*: $\rho$ parameter of *Dubins curve*;
4. *type_temp*: type of *word* in Dubins set $\mathcal{D}$;
5. *flag_temp*: it ca have value equal to 0 or 1.

The *edges.param* array is reported as an example in Table 3.6. As can be seen, the array starts from the $origin = [0, 0, 6.27]$ and for each new random point that respects the conditions defined above, we calculate the *Dubins curve* that has as extremes the $(i^{\text{th}} - 1)$ vertex (or random point) saved in the previous iteration and the $i^{\text{th}}$ random point of the $i^{\text{th}}$ iteration. For each *word* that forms the *Dubins curve*, the length of the curve is then calculated by summing the three parameters and then the one with the lowest sum is chosen. On the basis of this "cost", the *Dubins parameters* are gradually saved, starting from the closest vertex up to the first considered, the origin.

In Figure 3.20 it is represented the Dubins search tree. As can be seen, for random points generated near the position of the origin, the path is not coherent with the track layout defined by the cones. In fact, the trajectory, instead of growing along the x direction, circumscribes the first obstacles seen from Lidar sensor. Since the next

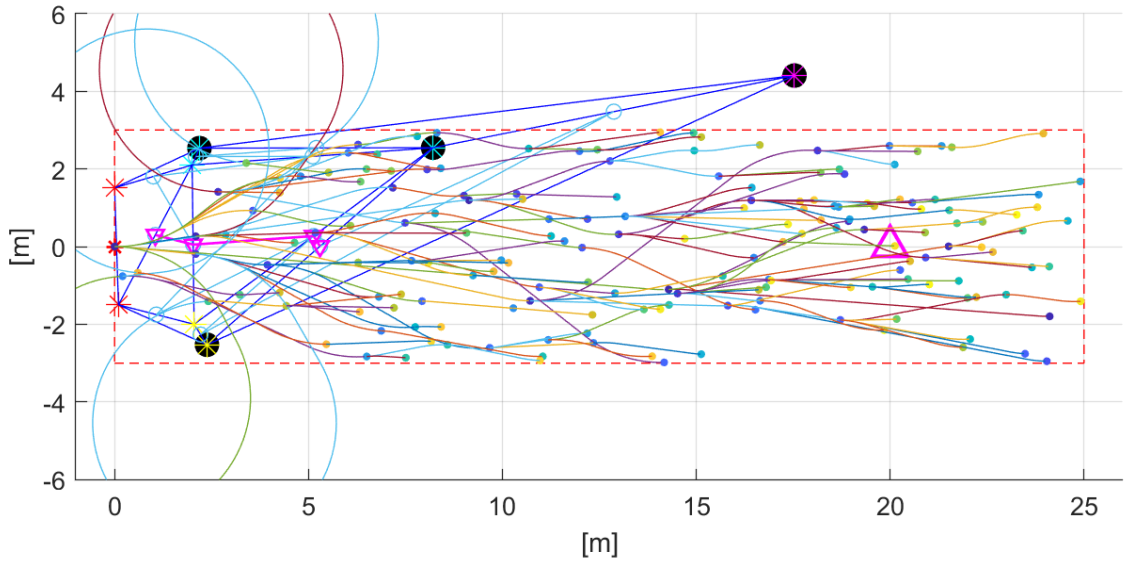



**Figure 3.20:** *Dubins curves search tree.*

step will be the search for the shortest trajectory, it may happen that a section of this trajectory is formed by a curve that circumscribes the cones instead of passing through them, therefore this is a wrong trajectory for our purposes. To avoid a path that has a circular section at the beginning, a further condition has been added for the generation of the random points, namely that each random point created must have a distance of at least 1 m from any other previously generated randomic point, Algorithm 9. In Figure

**Algorithm 9** Check minimum distances between generated randomic points

1: **procedure** CHECK_MINIMUM_DISTRAND($x_{chk}$, $y_{chk}$, $i$)
2: $count = i$;
3: **if** $(sqrt(((x_{chk}(count, 1) - 0)^2) + ((y_{chk}(count, 1) - 0)^2))) <= 1$ **then**
4: result1=0;
5: **return**
6: **end if**
7: **for** $e \leftarrow 1, (count - 1)$ **do**
8: **for** $m \leftarrow (e + 1), count$ **do**
9: **if** $(sqrt(((x_{chk}(e, 1) - x_{chk}(m, 1))^2) + ((y_{chk}(e, 1) - y_{chk}(m, 1))^2))) <= 1$ **then**
10: $result1 = 0$;
11: **end if**
12: **end for**
13: **end for**
14: $result1 = 1$;
15: **end procedure**

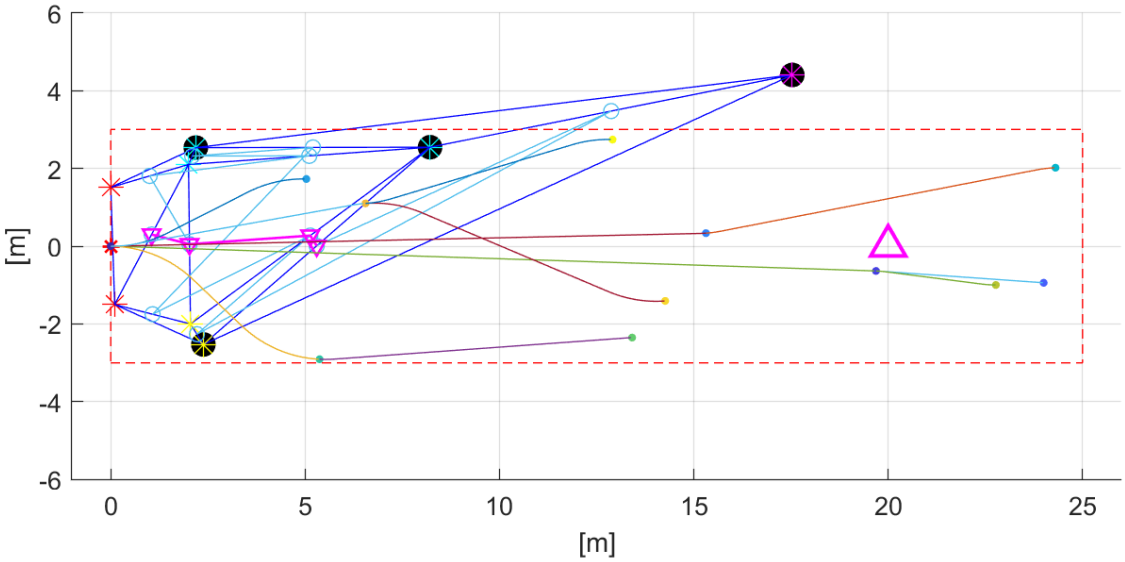
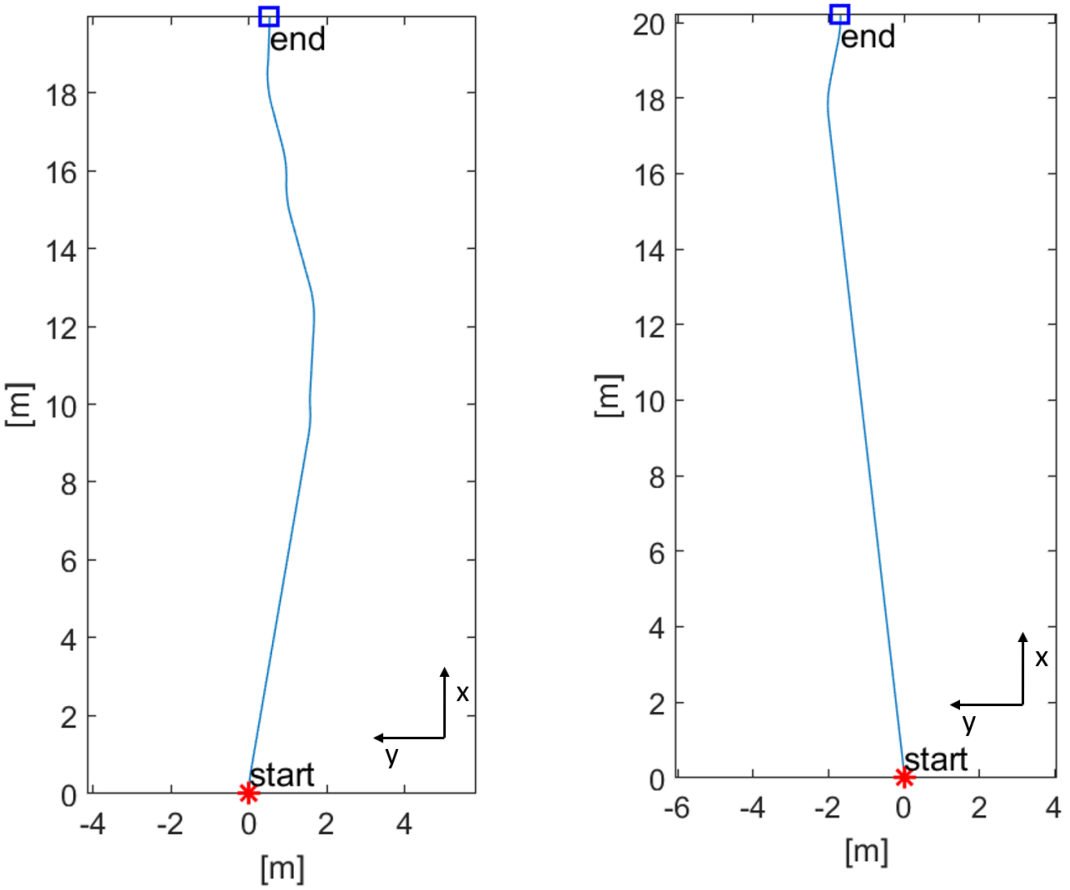



**Figure 3.21:** *Dubins curves search tree using Procedure descibed in Algorithm 9.*

3.21, the creation of the new *Dubins search tree* applying Algorithm 9 is described. The vertices that make up the tree are significantly lower respect to Figure 3.20, with the same number of iterations. The main positive aspect is that the probability of obtaining a path that forms a circle near the origin is significantly reduced. Secondly the path oscillations along the y direction are drastically attenuated. After the entire *Dubins search tree* is defined, it is possible to select the trajectory obviously choosing the one that has the shortest length. To select the best trajectory, this step is done backwards. First, the distance of each vertex of the tree with respect to the *finalGoal* of the specific frame is calculated. Given the list of all distances from the *finalGoal* to the respective node, the node with the shortest distance is chosen and that node is assigned as the final node. Gradually you look for the previous vertex connected to the next node by going backwards along the branch of the *Dubins tree* until the "stem" of the tree is reached, the origin. In Algorithm 10 is defined the procedure described above. In Figure 3.22a, the shortest path is shown with appreciable oscillations along the y direction. Applying the Procedure described with the Algorithm 9, the resulting shortest path, Figure 3.22b has a much more regular and fluid trend.

**Algorithm 10** Shortest path selection

```
1: procedure PATHSELECTION(vertecies, q_goal, ind_nearest_tree)
2:     for j ← 1, size(vertecies, 1) do
3:         tmpdist = sqrt((vertecies(j,1)−q_goal(1))^2+(vertecies(j,2)−q_goal(2))^2);
4:         D = [D tmpdist];
5:     end for
6:     [val, idx] = min(D);
7:     q_final = vertecies(idx, :);
8:     q_end = q_final;
9:     while FLAG_Traj == 0 do
10:        vertex_traj = [vertex_traj; vertecies(ind_nearest_tree(idx − 1), :)];
11:        traj.param = edges.param((idx − 1), :);
12:        if vertex_traj == origin then
13:            FLAG_Traj = 1;
14:        end if
15:    end while
16:    len_vertex = size(vertex_traj, 1);
17:    traj.vertex(1 : len_vertex, :) = vertex_traj;
18:    for i ← 1, (size(traj.vertex, 1) − 1) do
19:        start_param = traj.param(i, :)
20:        stepsize = 0.1;
21:        path1 = dubins_path_sample_many(start_param, stepsize);
22:        [path; flip(path1)];
23:    end for
24:    direction = ones(lenght(path), 1);
25: end procedure
```




**(a)** *Shortest path evaluation.*

**(b)** *Shortest path evaluation using Algorithm 9.*

**Figure 3.22:** *Dubins shortest path.*

### 3.2.3 Motion Planning model

After describing the operations of the *Local Path planner*, it can start the description of the *Motion Planning* model, that is the combination of the *Local Path planner* block, the *Controller* and the *Vehicle* model blocks. The Simulink™ model, shown in Figure 3.23, generates the steering, acceleration and deceleration commands that must be provided to the vehicle to complete the given *mission*. As explained in Paragraph 3.2, the acquisitions coming from the LiDAR pipeline, saved in an external HD, are supplied to the *Local Path planner* with a frequency of 10 Hz. Every 0.1 s the *Local Path planner* generates 2 outputs:

1. *path*: Poses of the path calculated by the *Local Path planner* ($< x, y, \theta >$), dimension [1000 $x$ 3];

2. *direction*: Direction of motion of the vehicle; vehicle forward motion ($< 1 >$), dimension [1000 $x$ 1].

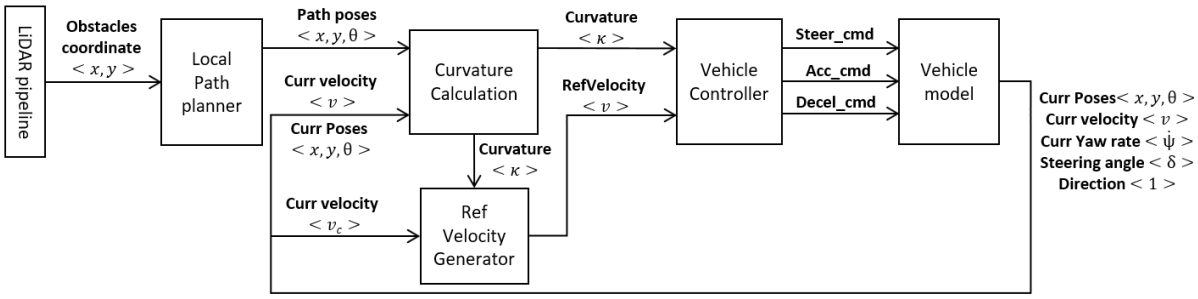



**Figure 3.23:** *Motion Planning model.*

#### Curvature calculation

Since the vehicle controller needs to receive as input the curvature of the trajectory to perform the control action on the steering angle, it needs to calculate from the *path* poses coming from the *Local Path planner* block the curvature values of the estimated trajectory. Therefore, given the series of points that make up the specific trajectory, the coefficients of a second degree polynomial that approximate the trend of this series of points are found. The *fitPolynomialRANSAC* MATLAB™ function is used for this

purpose; it exploits the M-estimator sample consensus (MSAC) algorithm, a variation of the random sample consensus (RANSAC) algorithm to fit the data [33]. The returned array, that includes the three coefficients $[a, b, c]$ of a second degree polynomial equation $ax^2 + bx + c$, with the additional information coming from the vehicle's current *pose* and *velocity*, are then used to compute the curvature with the following formula:

$$\kappa = \frac{|\dot{x}\ddot{y} - \ddot{x}\dot{y}|}{[(\dot{x}^2 + \dot{y}^2)]^{\frac{3}{2}}} \tag{3.30}$$

### Vehicle Controller

The control strategy for trajectory tracking is responsible of providing three input commands (acceleration, deceleration and steering command) to perform all the maneuvers autonomously in a safety manner. For the purpose of analysis it is deployed a decoupled control strategy to handle separately motions related to lateral and longitudinal dynamic. It was decided to use the geometric controller implemented in "Stanley", the Stanford Racing Team's car that won the DARPA Grand Challenge 2005 [18].

The vehicle model used for controller design is the *dynamic model*. Unlike the *kine-*

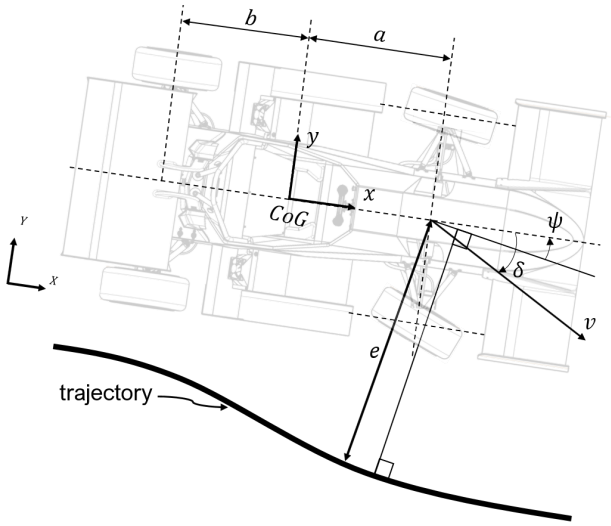



**Figure 3.24:** *Kinematic model for controller design.*

*matic model* which assumes the vehicle has negligible inertia, and therefore valid for low speed driving conditions, the *dynamic model* instead includes inertial effects such as tire slip and steering servo actuation. This more complicated, but more accurate,

scheme permits simulation for tuning and augmenting the controller to handle realistic dynamics [18]. The kinematic motion of a vehicle, with speed $v(t)$, can be described with the crosstrack error, $e(t)$, of the guiding wheels, and the angle of those wheels with respect to the nearest segment of trajectory to be tracked $(\psi(t) - \delta(t))$, as in Figure 3.24. The $\psi(t)$ is the yaw angle (heading) of the vehicle with respect to the closest trajectory segment, and $\delta(t)$ is the angle of the front wheels with respect to the vehicle. For forward driving, the guiding wheels are the front wheels, and the derivative of the crosstrack error is:

$$\dot{e}\left(t\right) = \nu\left(t\right)\sin\left(\psi\left(t\right) - \delta\left(t\right)\right) \tag{3.31}$$

where the steering is mechanically limited to $|\delta\left(t\right)| < \delta_{max}$. The derivative of the yaw angle, the yaw rate, is:

$$\dot{\psi}\left(t\right) = r\left(t\right) = -\frac{\nu\left(t\right)\sin\left(\delta\left(t\right)\right)}{a+b} \tag{3.32}$$

where $a$ and $b$ are the distance from the center of gravity (CoG) to the front and rear wheels, respectively.

In order to model the nonlinear dynamic motion of the vehicle, the effect of tire slip and of the steering servo motor are considered. The front and rear tire are modeled such that each provide a force $F_{yf}\left(t\right)$ and $F_{yr}\left(t\right)$, perpendicular to the rolling direction of the tire, and proportional to the *side slip* angle $\alpha\left(t\right)$. Reporting the *High-Speed cornering vehicle* model with the assumption of negligible vehicle track and considering only the contributions coming from the tires, it is obtained the *Monotrack* model [17],

$$\begin{aligned} F_{yf}\left(t\right) &\approx -C_y \alpha f\left(t\right) \\ F_{yr}\left(t\right) &\approx -C_y \alpha r\left(t\right) \end{aligned} \tag{3.33}$$

where $C_y$ refers to the Lateral stiffness of the tire, and

$$\begin{aligned} \alpha_f\left(t\right) &= \arctan\left(\frac{U_y\left(t\right) + r\left(t\right)a}{U_x\left(t\right)}\right) + \delta\left(t\right) \\ \alpha_r\left(t\right) &= \arctan\left(\frac{U_y\left(t\right) - r\left(t\right)b}{U_x\left(t\right)}\right) \end{aligned} \tag{3.34}$$

with body fixed longitudinal and lateral velocities, $U_x\left(t\right)$ and $U_y\left(t\right)$. The differential equations of motion associated to the *dynamic* model with the assumptions reported

before, Figure 3.25, are

$$\begin{aligned} m\left(\dot{U}_x(t) - r(t)\,U_y(t)\right) &= F_{xr} + F_{xf}\cos\delta(t) - F_{yf}\sin\delta(t) \\ m\left(\dot{U}_y(t) + r(t)\,U_x(t)\right) &= F_{yr} + F_{xf}\sin\delta(t) + F_{yf}\cos\delta(t) \\ I_z\dot{r}(t) &= aF_{xf}(t)\sin\delta(t) + aF_{yf}(t)\cos\delta(t) - bF_{yr}(t) \end{aligned} \tag{3.35}$$

where $F_{xf}(t)$ and $F_{xr}(t)$ are the components of the force provided by the front and

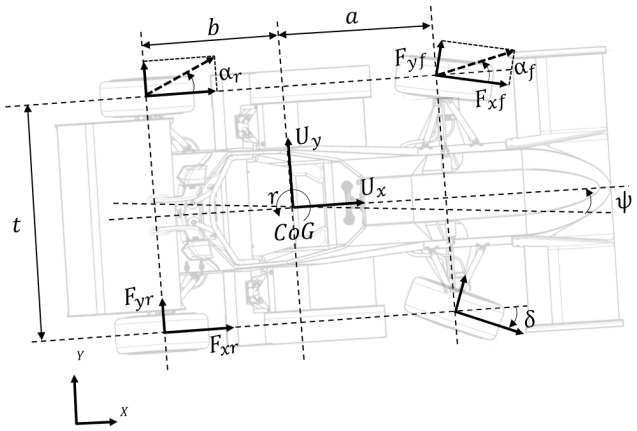



**Figure 3.25:** *Dynamic model.*

rear tires in their direction of rolling, while $F_{yf}(t)$ and $F_{yr}(t)$ the components of the force provided by the front and rear tires in lateral direction. The value of $C_y$ for the *SC 19* with race track tires was found to be 44 $\frac{\text{kN}}{\text{rad}}$.

### Tracking control laws - Lateral Control

This sections details the lateral control law. The controller takes the vehicle state and the commanded trajectory, and output commands at fixed rate, 10 Hz. The inputs used are:

1. *RefPose*: Reference location and orientation of the calculated trajectory;
2. *Curvature*: Curvature value extrapolation from the estimated path;
3. *Velocity*: Speed of the segment of trajectory closest to the front tires.

The controller is selected such that the resulting differential equation has a globally asymptotically stable equilibrium at zero crosstrack error. For the kinematic equations

of motion, given by Equation (3.31) and (3.32), the steering control law

$$\delta(t) = \begin{cases} \psi(t) + \arctan\frac{ke(t)}{\nu(t)} & if \quad \left|\psi(t) + \arctan\frac{ke(t)}{\nu(t)} < \delta_{max}\right| \\ \delta_{max} & if \quad \psi(t) + \arctan\frac{ke(t)}{\nu(t)} \geq \delta_{max} \\ -\delta_{max} & if \quad \psi(t) + \arctan\frac{ke(t)}{\nu(t)} \leq \delta_{max} \end{cases} \tag{3.36}$$

results in a closed loop system with a globally asymptotically stable equilibrium at $e = 0$ for $\nu(t) > 0$ and $0 < \delta_{max} < \frac{\pi}{2}$. Using the controller in Equation (3.36), the location of the front wheels is actively controlled, but the yaw is not [18]. The tire acting as dampers, provide reactions forces to sideways velocities. As speed increases, the damping effect diminishes, creating a need for active damping. This is done providing a negative feedback on yaw rate, without impacting tracking performance. Thus, $k_{d,yaw}(r_{traj}(t) - r_{meas}(t))$is added to the steering command, where $k_{d,yaw}$ is a tuned gain, $r_{traj}$ is the yaw rate for the trajectory, and $r_{means}$ is measured yaw rate.
In order to prevent overshoot that can cause instability in the servo steering mecchanism is added $k_{d,steer} = (\delta_{meas}(i) - \delta_{meas}(i+1))$ to the steering command, where $\delta_{meas}$ is the discrete time measurement of the steering angle, and $i$ is the index of the measurement one control period earlier.
Subsequently, it is found the steady state yaw, $\psi_{ss}$, relative to a constant curve path,

$$\psi_{ss} = \frac{m\nu(t)\, r_{traj}(t)}{C_y\left(1 + \frac{b}{a}\right)} = k_{ag}\nu(t)\, r_{traj}(t) \tag{3.37}$$

where $k_{ag} = \frac{m}{Cy + \left(1 + \frac{b}{a}\right)}$. Due to this non-zero yaw angle setpoint, the controller correctly turns the vehicle, achieving null crosstrack error. One final modification is done adding a tuned gain $k_{soft}$, so permitting soft control at low speeds. The *complete steering law*, compensating for dynamics, is

$$\boxed{\begin{aligned} \delta(t) = (\psi(t) - \psi_{ss}(t)) + \arctan\frac{ke(t)}{k_{soft} + \nu(t)} + k_{d,yaw}(r_{traj} - r_{meas}) + \\ + k_{d,steer}(\delta(i) - \delta(i+1)) \end{aligned}} \tag{3.38}$$

with saturation at $\pm\delta_{max}$. Table 3.7 shows the lateral controller variables values, tuned

**Table 3.7:** *Lateral controller Stanley SIMULINK™block settings.*

| **Lateral Controller Stanley** | |
|---|---|
| **Controller variable** | **Value** |
| Position gain of forward motion | 10^-5 |
| Yaw rate feedback gain ($k_{d,yaw}$) | 10^-20 |
| Steering angle feedback gain ($k_{d,steer}$) | 10^-20 |
| Vehicle mass (m) | $\approx$ 190 kg |
| Distance from CoG to front axle ($a$) | 0.839 m |
| Distance from CoG to rear axle ($b$) | 0.686 m |
| Front tire corner stiffness ($C_y$) | 44e3 $\frac{\text{N}}{\text{rad}}$ |
| Maximum steering angle ($\delta_{max}$) | 25 $^\circ$ |

for the *Acceleration* mission.

### Tracking control laws - Longitudinal Control

The longitudinal controller is responsible for the acceleration and deceleration command and uses as inputs:

1. *RefVelocity*: Reference velocity of the vehicle calculated on the basis of the curvature value extrapolated from the path of the *Local Path planner*;

2. CurrVelocity: Current Velocity of the vehicle;

3. *Direction*: Direction of motion of the vehicle.

The controller implemented for the longitudinal dynamic treats the brake cylinder pressure and the throttle level as two opposing, single-acting actuators that exert a longitudinal force on the car [18]. In our case the deceleration is directly done acting on the in-wheel electric motors that generate the torque to move the vehicle. The controller computes a single proportional integral (PI) error metric, at discrete control iteration $i+1$,

$$\boxed{e_\nu\left(i+1\right) = k_{p,\nu}\left(\nu\left(i+1\right) - \nu_c\left(i+1\right) + k_{i,\nu}e_{int}\left(i+1\right)\right)} \tag{3.39}$$

where the integral term is given by

$$e_{int}\left(i+1\right) = e_{int}\left(i\right) + \left(\nu\left(i+1\right) - \nu_c\left(i+1\right)\right) \tag{3.40}$$

**Table 3.8:** *Longitudinal controller Stanley SIMULINK™block settings.*

| **Longitudinal Controller Stanley** | |
|---|---|
| **Controller variable** | **Value** |
| Proportional gain ($K_p$) | 5 |
| Integral gain ($K_i$) | 1.5 |
| Sample time | 0.1 s |

and $\nu_c$ is the commanded speed. The values of $k_{p,\nu}$ and $k_{i,\nu}$ determine the trade-off between disturbance rejection and overshoot.
In particular, the *Longitudinal Controller Stanley* SIMULINK™ block implements a discrete proportional-integral (PI) controller, using the equation [35]:

$$u(k) = \left(K_p + K_i \frac{T_s z}{z-1}\right) e(k) \tag{3.41}$$

where

1. $u(k)$ is the control signal at the $k$th time step, the acceleration command *AccCmd* and deceleration *DecCmd* command;
2. $K_p$ is the proportional gain;
3. $K_i$ is the integral gain;
4. $T_s$ is the sample time of the block;
5. $e(k)$ is the velocity error at the $k$th time step. For each $k$, this error is equal to the difference between the current velocity and reference velocity inputs.

The block saturates the acceleration and deceleration commands setting appropriate *Maximum longitudinal acceleration* ($\frac{\mathrm{m}}{\mathrm{s}^2}$) and *Maximum longitudinal deceleration* value ($\frac{\mathrm{m}}{\mathrm{s}^2}$). The values, in this specific application are set $\pm$ 20 $\frac{\mathrm{m}}{\mathrm{s}^2}$. Table 3.8 shows the longitudinal controller variables values, tuned for the *Acceleration* mission.

### Vehicle Model - 3 DOF Single Track Model

The output commands, exiting from the vehicle controller, enter the vehicle model so the longitudinal, lateral and yaw motion is calculated. It is taken as reference the

*Vehicle Body 3DOF* SIMULINK™ block which implements a rigid two-axle vehicle model. For our purposes it is used the simplified *bicycle* model, where forces act along the center line at the front and rear axles and lateral load transfer is no taken into acocunt. Considering the *External longitudinal velocity* option in the *Axle Forces* setting, the block assumes that the external longitudinal velocity is in quasi-steady state, so the longitudinal acceleration is approximately zero. Since the motion is quasi-steady, the block calculates lateral forces using the tire slip angles and linear cornering stiffness. This setting option is choosen since, at this stage, driveline and nonlinear tire responses are non considered for the description of the vehicle motion.
The block uses the equations reported in [34], following the reference system shown in Figure 3.26,

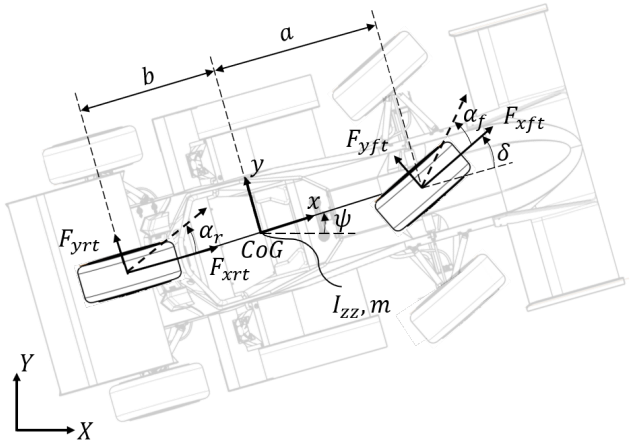



**Figure 3.26:** *3 DOF Single Track Model.*

$$\begin{aligned} \ddot{x} &= \dot{y}r + \frac{F_{xf} + F_{xr} + F_{x,ext}}{m} \\ \ddot{y} &= -\dot{x}r + \frac{F_{yf} + F_{yr} + F_{y,ext}}{m} \\ \dot{r} &= \frac{aF_{yf} - bF_{yr} + M_{z,ext}}{I_{zz}} \\ r &= \dot{\psi} \end{aligned} \tag{3.42}$$

where $r$ is the yaw rate, being $\psi$ the yaw angle. As external forces, both drag and external force inputs can be calculated, the latter considering they act on the CoG of

the model.

$$\begin{aligned} F_{xyz,ext} &= F_{d,xyz} + F_{input,xyz} \\ M_{xyz,ext} &= M_{d,xyz} + M_{input,xyz} \end{aligned} \tag{3.43}$$

Longitudinal and lateral tire force applied to front an rear wheels respectively follow the equations:

$$\begin{aligned} F_{xft} &= F_{xfinput} \\ F_{yft} &= -C_{yf}\alpha_f\mu_f\frac{F_{zf}}{F_{z,nom}} \\ F_{xrt} &= 0 \\ F_{yrt} &= -C_{yr}\alpha_r\mu_r\frac{F_{zr}}{F_{z,nom}} \end{aligned} \tag{3.44}$$

where $F_{xfinput}$ is the input force applied to the vehicle CoG, along the x-axis, $C_{yf}$ and $C_{yr}$ are the front and rear tire cornering stiffness, $\alpha_f$ and $\alpha_r$ are the front and rear wheel side slip angles, $\mu_f$ and $\mu_r$ are front and rear wheel friction coefficient, $F_{zf}$ and $F_{zr}$ are the normal force applied to front and rear wheels, along the vehicle-fixed z-axis and $F_{z,nom}$ is the nominal normal force applied to axles, along the vehicle-fixed z-axis. To calculate the normal forces applied to front and rear tires, $F_{zf}$ and $F_{zr}$, it is used the pitch and roll equilibrium:

$$\begin{aligned} F_{zf} &= \frac{bmg - (\ddot{x} - \dot{y}r)\,mh + hF_{x,ext} + bF_{z,ext} - M_{y,ext}}{a+b} \\ F_{zr} &= \frac{amg + (\ddot{x} - \dot{y}r)\,mh + hF_{x,ext} + aF_{z,ext} - M_{y,ext}}{a+b} \end{aligned} \tag{3.45}$$

where $a$, $b$ and $h$ are geometric values of the model, representing the distances from front and real wheels to the vehicle CoG, and the height of the vehicle CoG from ground respectively. $F_{x,ext}$, $F_{z,ext}$ and $M_{y,ext}$ are the exteral forces and moments applied to the vehicle CoG along the vehicle-fixed x-, z- and y- axis.

To determine the front and rear tire slip angles, similarly to Equations 3.34,

$$\begin{aligned} \alpha_f &= \arctan\left(\frac{\dot{y} + ar}{\dot{x}}\right) - \delta_f \\ \alpha_r &= \arctan\left(\frac{\dot{y} + br}{\dot{x}}\right) - \delta_r \end{aligned} \tag{3.46}$$

Once determined the front and rear side slip angles value, the tire forces are calculated:

$$\begin{aligned} F_{xf} &= F_{xft}\cos(\delta_f) - F_{yft}\sin(\delta_f) \\ F_{yf} &= -F_{xft}\sin(\delta_f) + F_{yft}\cos(\delta_f) \\ F_{xr} &= F_{xrt}\cos(\delta_r) - F_{yrt}\sin(\delta_r) \\ F_{yr} &= F_{xrt}\sin(\delta_r) + F_{yrt}\cos(\delta_r) \end{aligned} \tag{3.47}$$

For what concern the aerodynamic drag, firstly the block transforms the wind speeds from the inertial frame to the vehicle-fixed frame:

$$\begin{aligned} w_x &= W_x\cos(\psi) + W_y(\psi) \\ w_y &= W_y\cos(\psi) - W_x(\psi) \\ w_z &= W_z \end{aligned} \tag{3.48}$$

Then it is determined the relative airspeed, subtracting the wind speed from the CoG vehicle velocity:

$$\bar{w} = \sqrt{(\dot{x}_b - w_x)^2 + (\dot{x}_y - w_x)^2 + w_z^2} \tag{3.49}$$

Using the airspeed, the block determines the drag forces:

$$\begin{aligned} F_{dx} &= -\frac{1}{2TR}C_d A_f P_{abs}\left(\bar{w}^2\right) \\ F_{dy} &= -\frac{1}{2TR}C_s A_f P_{abs}\left(\bar{w}^2\right) \\ F_{dz} &= -\frac{1}{2TR}C_l A_f P_{abs}\left(\bar{w}^2\right) \end{aligned} \tag{3.50}$$

where $C_d$, $C_s$ and$C_l$ are the air drag coefficients acting along vehicle fixed x- y- and z-axis. $T$ is the environmental air temperature, $R$ is the atmospheric specific gas constant $P_{abs}$ is the environmental absolute pressure and $A_f$ represents the vehicle frontal area.

**Table 3.9:** *Vehicle body 3 DOF SIMULINK™block settings.*

| **Vehicle body 3 DOF** | |
|---|---|
| **Vehicle parameters** | **Value** |
| Vehicle mass ($m$) | $\approx$ 190 kg |
| Longitudinal distance from CoG to front axle ($a$) | 0.839 m |
| Longitudinal distance from CoG to rear axle ($b$) | 0.686 m |
| Vertical distance from CoG to axle plane ($h$) | 0,242 m |
| Initial inertial frame longitudinal position ($X_o$) | 0 m |
| Initial longitudinal velocity ($x_o$) | 0 $\frac{\text{m}}{\text{s}}$ |
| Front\Rear tire stiffness ($C_{yf}C_{yr}$) | 44e3 $\frac{\text{N}}{\text{rad}}$ |
| Yaw polar inertia ($I_{zz}$) | 318.48 kgm$^2$ |
| Longitudinal drag area ($A_f$) | 2 m$^2$ |
| Longitudinal drag coefficient ($C_d$) | 0.3 |
| Longitudinal lift coefficient ($C_l$) | 0.1 |
| Longitudinal drag pitch moment ($C_{pm}$) | 0.1 |
| Relative wind angle vector ($\beta_w$) | [0:0.01:0.3] rad |
| Side force coefficient vector ($C_s$) | [0:0.03:0.9] |
| Yaw moment coefficient vector ($C_{ym}$) | [0:0.01:0.3] |
| Absolute pressure ($P_{abs}$) | 101325 Pa |
| Air Temperature ($T$) | 273 K |
| Nominal normal force ($F_{z,nom}$) | 5000 N |

Subsequently, it follows the calculation of the drag moments:

$$\begin{aligned} M_{dr} &= -\frac{1}{2TR} C_{rm} A_f P_{abs} \left(\bar{w}^2\right) (a+b) \\ M_{dp} &= -\frac{1}{2TR} C_{pm} A_f P_{abs} \left(\bar{w}^2\right) (a+b) \\ M_{dy} &= -\frac{1}{2TR} C_{ym} A_f P_{abs} \left(\bar{w}^2\right) (a+b) \end{aligned} \tag{3.51}$$

where $C_{rm}$ is the air drag roll moment acting about the vehicle fixed x-axis, $C_{pm}$ is the air drag pitch moment acting about the vehicle fixed y-axis and $C_{ym}$ represents the air drag yaw moment acting about the vehicle fixed z-axis. Table 3.9 shows the vehicle model parameters entered in the *Vehicle body 3 DOF* SIMULINK™ block.

### Reference velocity generator

In order to determine the reference speed profile, it is calculated the maximum speed using the curvature of the road, given by:

$$\nu_{max} = \sqrt{\frac{g\mu}{\kappa}} \tag{3.52}$$

where $\mu$ is the tire friction coefficient and $\kappa$ is the curvature value extrapolated from the "Curvature Calculation" block, Figure 3.23. The reference speed profile is constructed following a trapezoidal trend. Based on the position traveled by the vehicle, the speed increases following the Equation (3.52). After reaching about 80 m, the speed profile begins to decrease until it reaches zero. In this case, at the target speed of 33.3 $ms^{-1}$, which corresponds to the maximum vehicle speed setting an output power of 70 kW, the reference speed is obtained subtracting the target speed from the reference speed value obtained from Equation (3.52). In this way the vehicle is able to carry out the acceleration test in 75 m and to stop in the remaining 100 m in total safety, eliminating the task of perceiving the orange cones from the Stereo camera sensor and start the deceleration phase once the camera detects the orange color on an ongoing basis; thus knowing only the space traveled by the vehicle is a safer and more reliable approach. 5 m of tolerance have been taken as it is preferable to cut the finish line at maximum speed and because from simulations, considering low conditions of grip, it is possible to brake the vehicle in about 20 m, much smaller stopping distance respect to the limit imposed in the FSG Rulebook [39]. It is relevant to remember that by Regulations, it is not possible to stop the vehicle by activating the RES and therefore operating the EBS but rather generating the brake torque through the Service Brake System, which in our case is done by the regenerative braking implemented directly by the in-wheel electric motors. In fact, once reached a speed value equal to zero, the vehicle must enter in the *AS Finished* state and not the *AS Emergency* state, as prescribed in rule D.5.3.4 in [39].

Chapter 4

# Results and Discussion

In this Chapter, the simulation results of the *Acceleration* mission are presented. A detailed discussion about the output commands and other key paramaters in the description of the selected mission is also reported.

## 4.1 Data Collection

The simulation environment is built using MATLAB™ and SIMULINK™ software. The entire system composed by *Path Planning* and *Control* blocks receives inputs from the LiDAR sensor pipeline acquisitions, Figure 3.23. In the following sections two *Scenarios* representing the *Acceleration* mission are reported. The first one is a data collection done in September 2020 with a track layout that recreates the *Trackdrive* mission. Overall, the cones layout replicate an entire circuit, but it is sufficient to post process the initial 150 frames representing the straight section of the circuit, useful to recreate the *Acceleration* mission.

The second *Scenario* is an acquisitions done in December 2020. In this case it was chosen to build a race environment in which the track width is sligthly lower respect to the previous *Scenario* and the cones on the same side of the track boundaries are positioned closer to each other. This changes in the track layout has done in order to increase the number of cones per frame perceived by the LiDAR sensor.

**(a)** *Data collection in Cerrina racetrack (Sept., 2020).*

**(b)** *Data collection in Aeroclub Torino (Dec., 2020).*

**Figure 4.1:** *LiDAR sensor data acquisitions for the Acceleration mission.*

## 4.2 Scenario 1 - Acquisition Sept2020

The simulation results of the acquisition done at the Cerrina racetrack in September 2020 is presented in this section. As can be seen in Figure 4.1a, the first section of the circuit defined by the cones is the zone taken in account so as to recreate the simulation environment. In fact, the initial straight part is used to recreate the *Acceleration* mission. In this *Scenario*, it is tested the critical condition in which the track width defined by the cones is very wide, about 6 m. It must be remembered that by Regulations [39], the minimum track width is 3 m and from the experience acquired by the Squadra Corse FS team in FS competitions, the racing fields never have a track width recreated in this *Scenario*. However, even in the critical conditions in which the LiDAR sensor perceives from 1 to 3 obstacles per frame, the SIMULINK™ model responds correctly. The perception critical issues from the LiDAR sensor are overcome adding the array of four points on the sides of the racecar per each frame, as fully explained in Paragraph 3.2.1. In fact, in this way it is possible to discretize the surrounding space with good reliability, and ultimately to calculate the path the vehicle has to follow.

Firstly the three commands that come out from the *Vehicle Controller* are reported, namely the acceleration, deceleration and steering command. The *Acc/Dec* command is expressed in $\frac{\mathrm{m}}{\mathrm{s}^2}$ while the *Steer* command is indicated in °. For what concern the

g-force during the *Acceleration* mission, the vehicle achieves a maximum longitudinal acceleration of about 2.04 g.

For what concerns the steering command, the controller requires small variation of steering angle (tenths of degree) in accordance to the specific mission. The steering command varies between $+0.4$ ° and $-0.8$ °, values consistent with the correct functioning of the steering actuator.

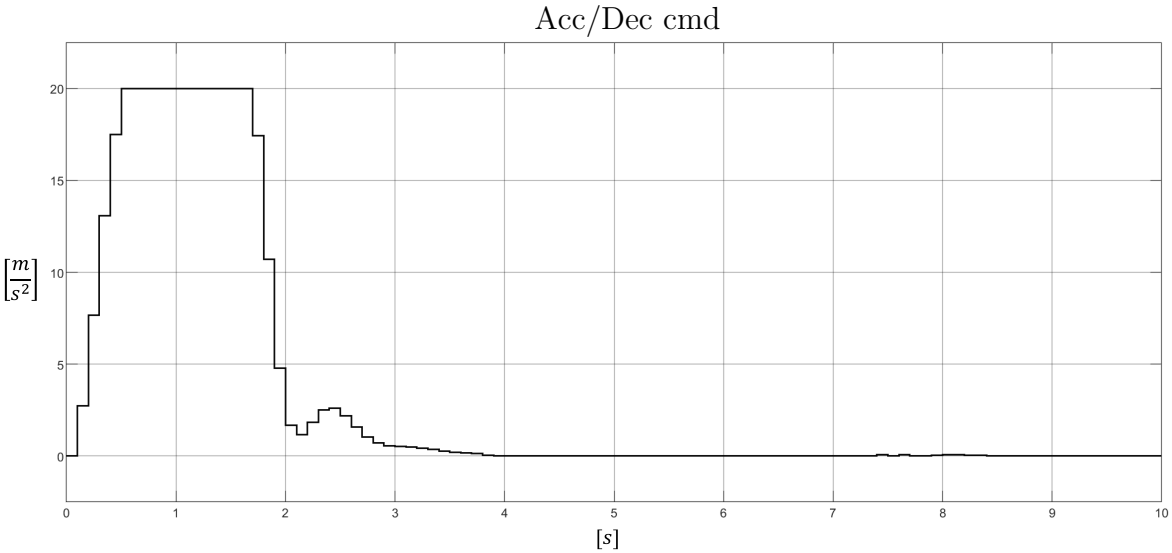



**Figure 4.2:** *Acceleration/Deceleration command.*

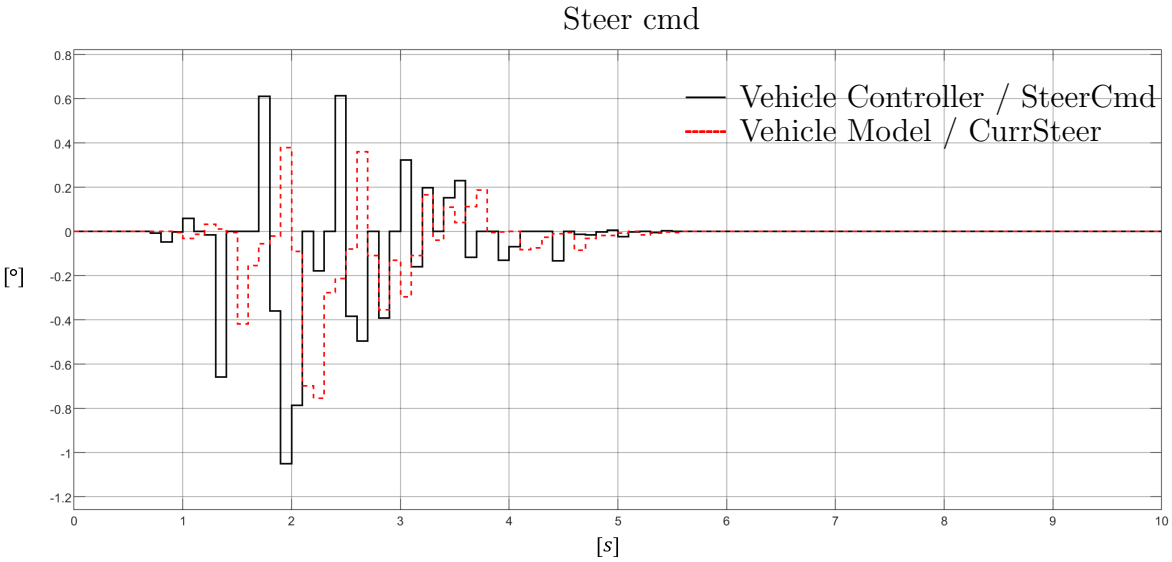



**Figure 4.3:** *Vehicle motion during the acceleration mission.*

Regarding the speed profile, the trapezoidal profile is reported in Figure 4.4. The reference speed is calculated using the Equation (3.52) and following the procedure described in Paragraph 3.2.3. As can be seen the reference speed reaches the target speed of 33,3 $\frac{\text{m}}{\text{s}}$ that is the maximum vehicle speed achievable by the racecar. With proper setting of proportional and integral gain ($K_p$ and $K_i$) of the *Longitudinal Stanley Controller* block, the *input* and subsequent *current* velocity values follow the reference speed trend.

In Figure 4.5, the vehicle space covered during the *Acceleration* mission is reported.

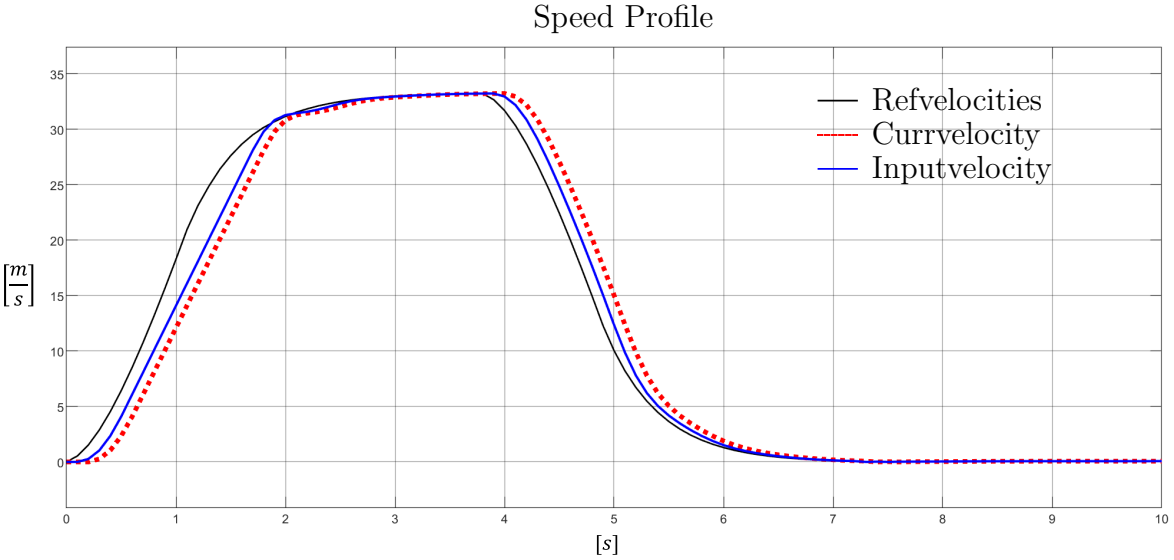



**Figure 4.4:** *Trapezoidal speed profile. Reference, input and current velocities are represented.*

The vehicle is able to finish the mission in 3.573 s. The vehicle is also capable of stopping in less than 125 m, remembering that by Regulations, the total distance that the vehicle can travel, considering from when the vehicle starts until it stops, is 175 m [39].

The vehicle yaw rate is reported in Figure 4.12. As can be seen, the variation of the yaw angle is appreciable when the control generates a variation on the steering command represented in Figure 4.9.

As can be seen from the curvature values in Figure 4.7, the paths generated by the *Local Path planner* with a frequency rate of 10 Hz are quite straight. In the initial phase, a series of oscillations can be noticed, which therefore determine the variation of tenths of a degree on the steering angle command by the vehicle controller. Differently from the *Scenario* 2, the curvature here has slightly greater values. This is mainly due to the fact that the track width, in this *Scenario*, is wider than in the next one and

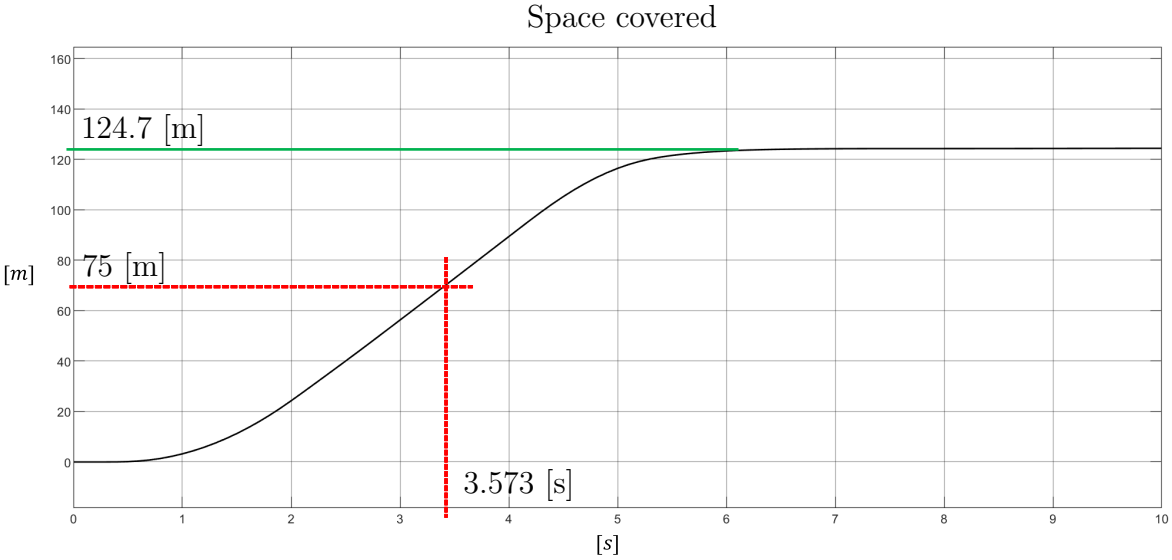



**Figure 4.5:** *Vehicle motion along the x-axis during the Acceleration mission.*

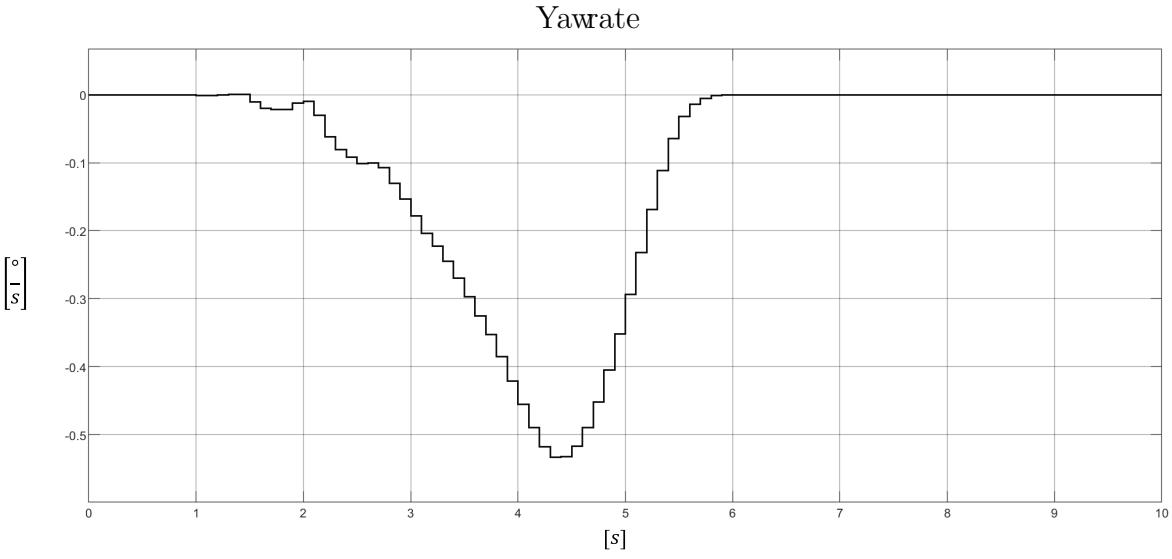



**Figure 4.6:** *Yaw rate trend during the Acceleration mission.*

above all the number of perceived obstacles per frame are lower, thus determining a less binding *Search Tree* expansion in the RRT Algorithm and therefore a path more subjected to possible oscillations.

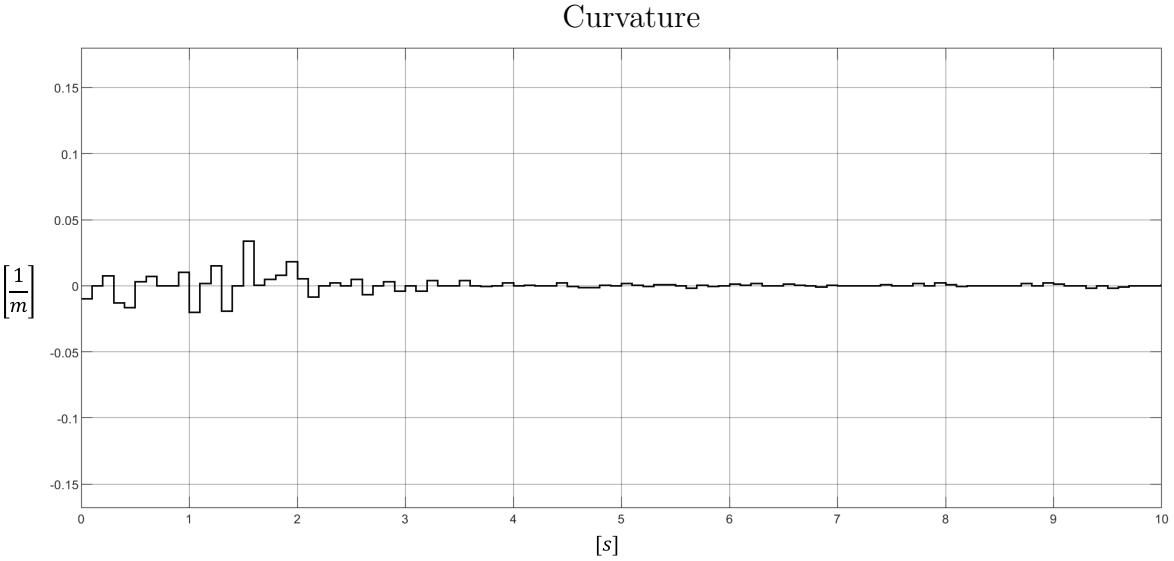



**Figure 4.7:** *Curvature extrapolation value for each frame.*

## 4.3 Scenario 2 - Acquisition Dec2020

The simulation results of the acquisition done at the Aeroclub Torino landing strip in December 2020 is presented in this section. In contrast to what already seen in the previous *Scenario*, the LiDAR sensor is able to determine a greater number of obstacles per frame. This is essentially due to the fact that the cones on the same side of the carriageway are positioned at shorter distance than in the previous *Scenario* but always within the limit imposed by the Regulations [39], that is 5 m. In addition, the opposing cones that define the borders of the track environment are placed so that the track width is smaller respect to the previous *Scenario*. The greater number of perceived obstacles helps the construction of the *Delaunay Triangulation*, since by increasing the number of vertices (perceived obstacles), the space is better discretized and the *Local Path planner* algorithm generate a more reliable and coeherent goal with the racetrack environment, as already explained in Paragraph 3.2.1.

Firstly the three commands that come out from the *Vehicle Controller* are reported, namely the acceleration, deceleration and steering command. The *Acc/Dec* command

is expressed in $\frac{\mathrm{m}}{\mathrm{s}^2}$ while the *Steer* command is indicated in °. For what concern the g-force during the *Acceleration* mission, the vehicle achieves a maximum longitudinal acceleration of about 2.04 g. In racing conditions, with not too hot tarmac temperature, by setting the vehicle output power at maximum (about 70 kW) as prescribed by [39], a new tire set is able to withstand that longitudinal acceleration, avoiding skidding when the vehicle starts from standstill, thus decreasing the time to travel 75 m during the *Acceleration* mission.

For what concerns the steering command, the controller requires small variation of steering angle (tenths of degree) in accordance to the specific mission. The *Steer cmd* saturation is set at 25° with $\delta_{max}$, Table 3.7.

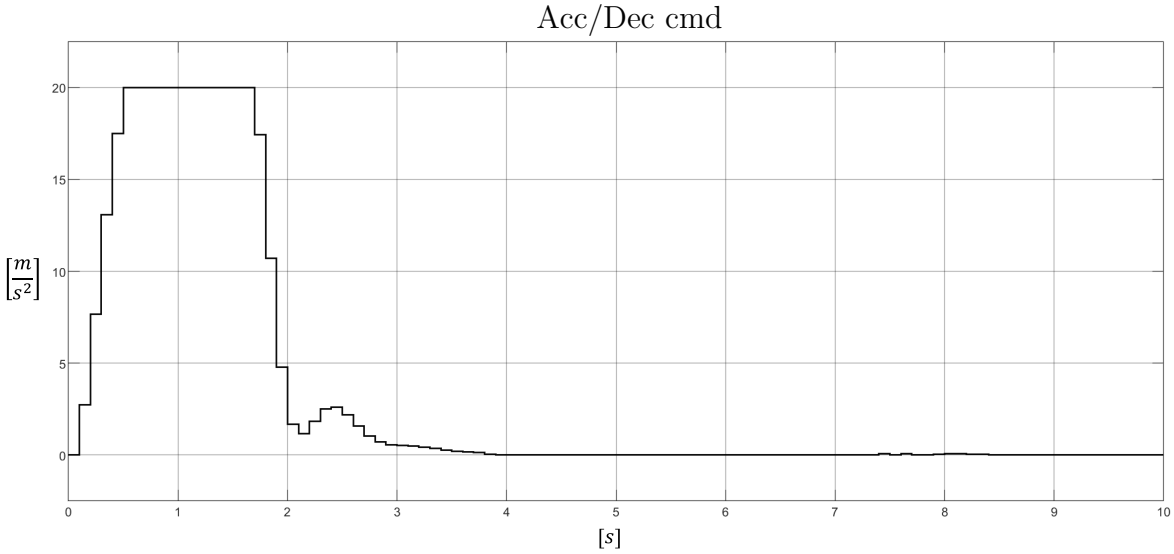



**Figure 4.8:** *Acceleration/Deceleration command.*

Regarding the speed profile, the trapezoidal profile is reported in Figure 4.10. The reference speed is calculated using the Equation (3.52) and following the procedure described in Paragraph 3.2.3. As can be seen the reference speed reaches the target speed of 33,3 $\frac{\mathrm{m}}{\mathrm{s}}$ that is the maximum vehicle speed achievable by the racecar. With proper setting of proportional and integral gain ($K_p$ and $K_i$) of the *Longitudinal Stanley Controller* block, the *input* and subsequent *current* velocity values follow the reference speed trend.

In Figure 4.11, the vehicle space covered during the *Acceleration* mission is reported.

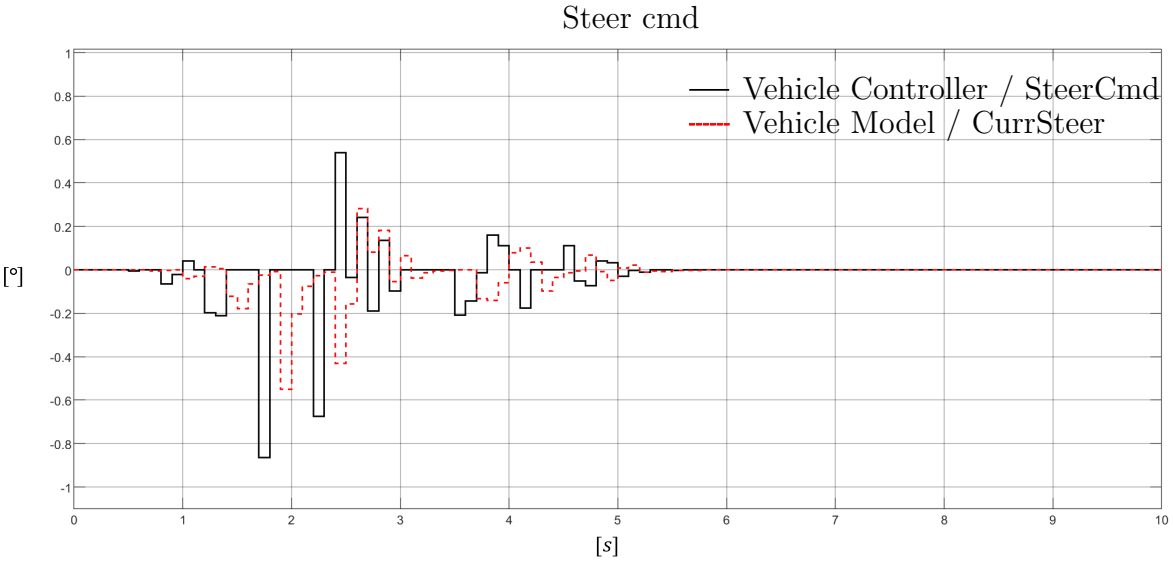



**Figure 4.9:** *Vehicle motion during the acceleration mission.*

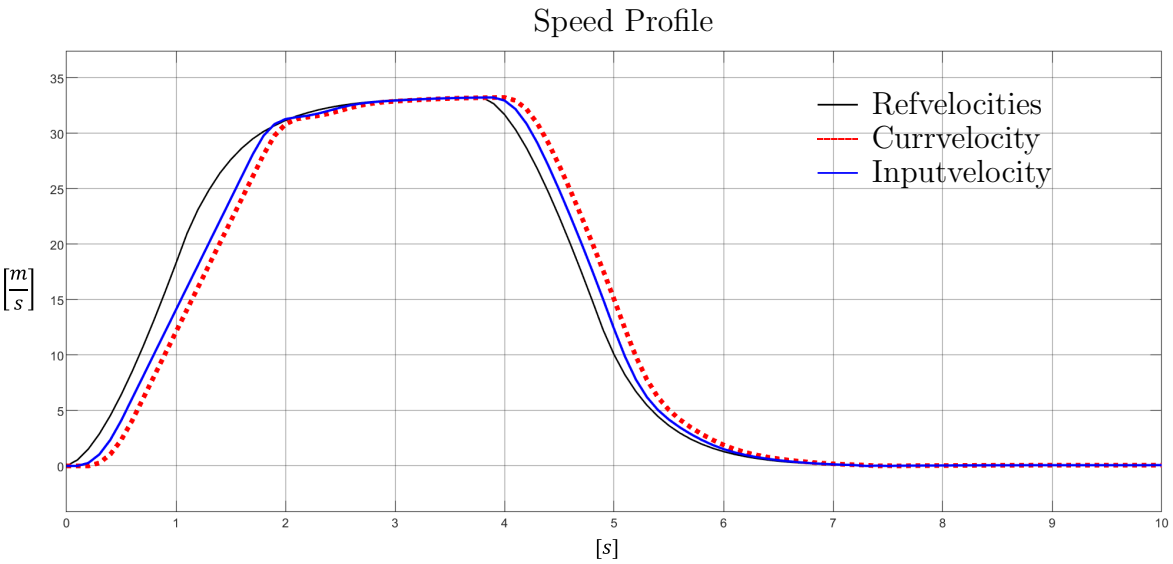



**Figure 4.10:** *Trapezoidal speed profile. Reference, input and current velocities are represented.*

The vehicle is able to finish the mission in 3.566 s, potentially finishing in first position, with a gap of 30 ms from the ETH Driverless Team, winners of the *Acceleration* mission in 2019 during the FSG competition, Table 2.1. The vehicle is also capable of stopping in 124 m, remembering that by Regulations, the total distance that the vehicle can travel, considering from when the vehicle starts until it stops, is 175 m [39].

The vehicle yaw rate is reported in Figure 4.12. As can be seen, the variation of

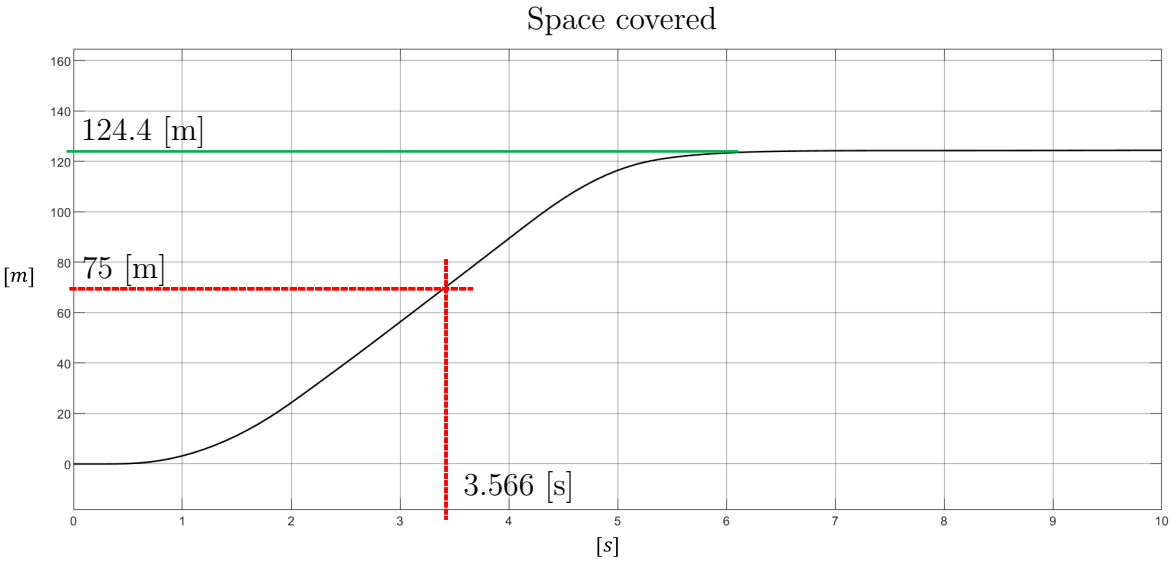



**Figure 4.11:** *Vehicle motion along the x-axis during the Acceleration mission.*

the yaw angle is appreciable when the control generates a variation on the steering command represented in Figure 4.9.

Finally, as can be seen from the curvature values in Figure 4.13, the paths generated by the *Local Path planner* with a frequency rate of 10 Hz are quite straight. In the initial phase a series of oscillations can be noticed, which therefore determine the variation of tenths of a degree on the steering angle command by the vehicle controller.

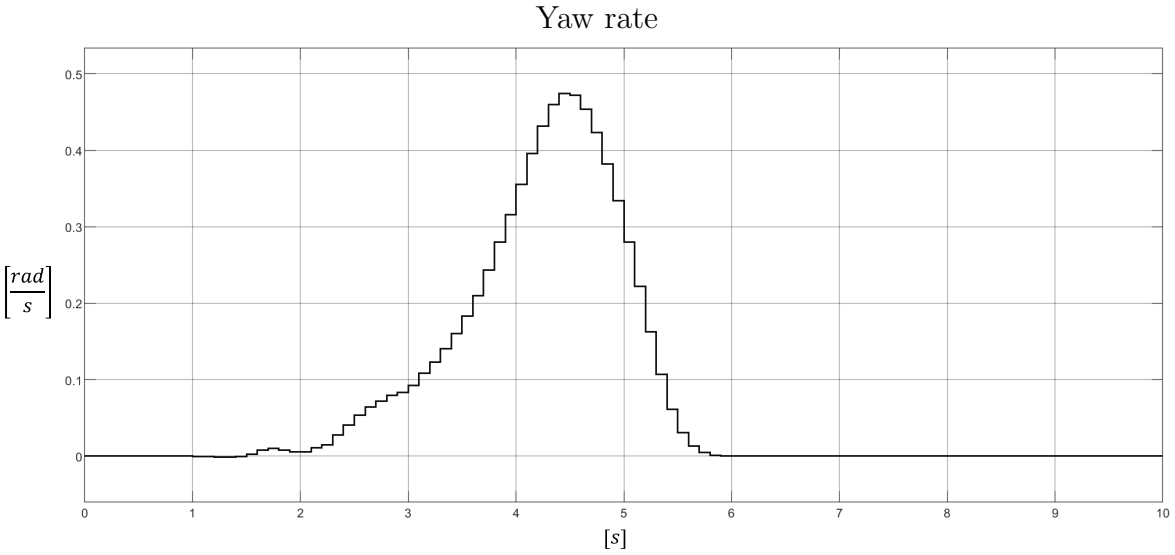



**Figure 4.12:** *Yaw rate trend during the Acceleration mission.*

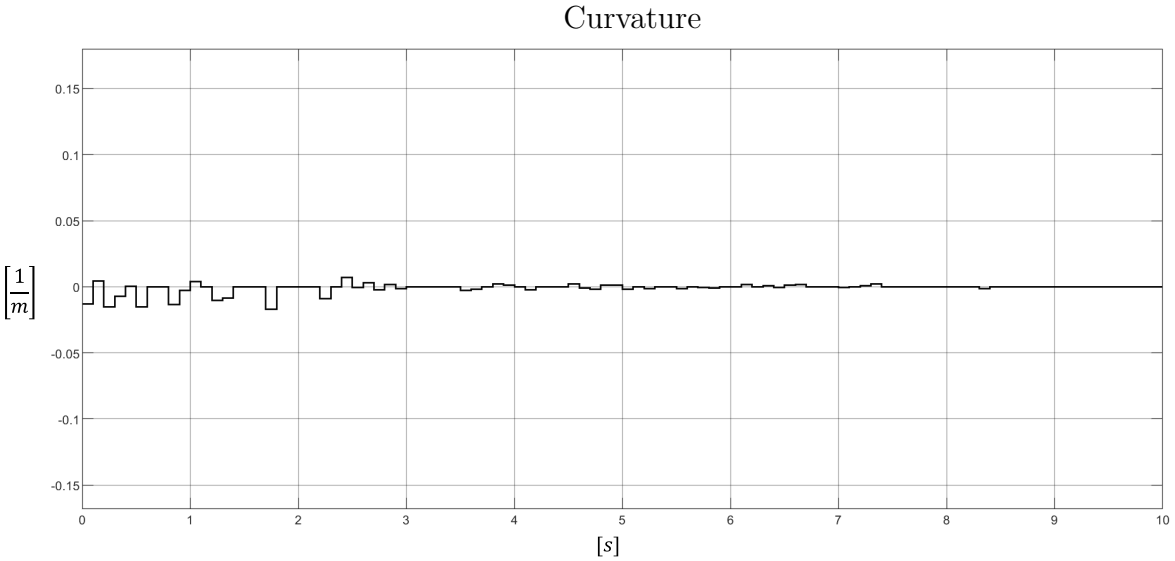



**Figure 4.13:** *Curvature extrapolation value for each frame.*

Chapter 5

# Conclusions and Future Works

In this experimental work, the *Motion Planning* problem for an autonomous racecar has been extensively analyzed. The *Local Path planner* was designed using of the *Dubins curves* applied to an RRT algorithm. First of all, it has been described the logical process that leads to the automatic definition of the goal per each frame. Subsequently, the controller and the vehicle model were specified in order to be able to consistently define the three output commands that manage the motion of the vehicle.

For what concern the *Acceleration* mission, to validate the model implemented on SIMULINK™ software, it was necessary to acquire the data coming from both the LiDAR and the Stereo camera sensor. For the specific mission, it is trivial to post-process the data coming from the Stereo camera sensor since the vehicle has to go only in straight direction for 75 m. Therefore, it was decided to collect only the data coming from the LiDAR sensor pipeline, as reported in Figure 3.7. As can be seen from the graphs shown in Paragraph 4, the Model, even if subjected to competition environments that could differ from each other, as regarded in the two *Scenarios* for what concerns the track width or the distance between two cones belonging to the same track side, behaves almost the same. For this mission, the track layouts geometric differences do not affect the behavior of the vehicle and above all the time to cover 75 m. In fact, even if the curvature, yaw rate and steering command values differ slightly between each *Scenario* and simulation due to the randomic *Search Tree* expansion, the speed profile and the distance traveled have a trend that is always similar to the previous

simulation, traveling 75 m in a time that would position the SC 19 certainly on the podium.

In the future evolution of this project, it will be necessary to integrate the *Local Path planner* and the *Vehicle Controller* on the dSpace™ MicroAutobox, a real-time system for performing fast function prototyping, in other words the ECU of the SC19 racecar. In addition, it will also be necessary to mount the steering actuator and the emergency braking system (EBS) onboard the racecar, as well as implement the state machine with different states depending on the mission the vehicle has to carry out.

For what concerns the *Trackdrive* mission, the *Local Path planner* currently uses only the data coming from the LiDAR sensor. In the condition in which the LiDAR sensor perceives a left or a right curve, in about 23 % of the total frames analyzed (about 4000), the *Local Path planner* defines a goal that is on the side of the track and therefore the generated path is not coherent with the real layout of the track, moving the vehicle in a misleading direction. This is not a big deal considering the fact that every 0.1 s, the trajectory is updated by the *Local Path planner* but nevertheless this issue leads to dynamic oscillations and vehicle instabilities that it is preferible to avoid. For this reason, the intention is to combine the map generated by the LiDAR sensor with the information on the color of the cones coming from the Stereo camera sensor pipeline. In this case, the error in the automatic goal generation could be considerably reduced, by inserting as a validity condition for the extrapolation of potential goals (PotGoal, Paragraph 3.2.1, only those points that are located on segments that have as extremes cones with different colors, thus minimizing the possibility of identifiyng points on the side of the track.

After the vehicle has covered a whole lap of the track, returning to a position already traveled previously, the *Motion Planning* controller must be changed, using a *Model predictive control*, since the track will be known after the first lap and the aim at that point will be to have an aggressive drive, trying to improve the lap time at every loop. Therefore, it will be necessary to integrate the *Local Path planner* with the MPC for the laps following the first one. In fact, at the beginning, the racecar will investigate the surrounding environment at very low speed, using for this purpose, the Stanley controller presented in [18]. Once the racecar has managed autonomously to cut again

the finish line, throught the information from the GPS sensor, the state machine will have to select the *Motion Planning* model with the *Model Predictive Controller*.

# List of Figures

# List of Tables

# Bibliography


[1] Ahmad Abbadi and Radomil Matousek. Path planning implementation using matlab. *Technical Computing Bratislava*, pages 1–5, 2014.

[2] Ahmad Abbadi, Radomil Matousek, Pavel Osmera, and Lukas Knispel. Spatial guidance to rrt planner using cell-decomposition algorithm. In *20th international conference on soft computing, MENDEL*, volume 2014, 2014.

[3] Ahmad Abbadi and Václav Přenosil. Safe path planning using cell decomposition approximation. *Distance Learning, Simulation and Communication*, 8:2–3, 2015.

[4] Jérôme Barraquand and J-C Latombe. A monte-carlo algorithm for path planning with many degrees of freedom. In *Proceedings., IEEE International Conference on Robotics and Automation*, pages 1712–1717. IEEE, 1990.

[5] Martin Buehler, Karl Iagnemma, and Sanjiv Singh. *The 2005 DARPA grand challenge: the great robot race*, volume 36. Springer, 2007.

[6] Martin Buehler, Karl Iagnemma, and Sanjiv Singh. *The DARPA urban challenge: autonomous vehicles in city traffic*, volume 56. springer, 2009.

[7] John Canny, Bruce Randall Donald, John Reif, and Patrick G Xavier. On the complexity of kinodynamic planning. Technical report, Cornell University, 1988.

[8] Howie M Choset, Seth Hutchinson, Kevin M Lynch, George Kantor, Wolfram Burgard, Lydia E Kavraki, Sebastian Thrun, and Ronald C Arkin. *Principles of robot motion: theory, algorithms, and implementation*. MIT press, 2005.

[9] European Commission. Communication from the commission to the european parliament, the council, the european economic and social committee and the committee of the regions youth opportunities initiative. 2011.

[10] Boris Delaunay et al. Sur la sphere vide. *Izv. Akad. Nauk SSSR, Otdelenie Matematicheskii i Estestvennyka Nauk*, 7(793-800):1–2, 1934.

[11] Bruce R Donald and Patrick G Xavier. Provably good approximation algorithms for optimal kinodynamic planning for cartesian robots and open chain manipulators. In *Proceedings of the sixth annual symposium on Computational geometry*, pages 290–300, 1990.

[12] Lester E Dubins. On curves of minimal length with a constraint on average curvature, and with prescribed initial and terminal positions and tangents. *American Journal of mathematics*, 79(3):497–516, 1957.

[13] Thierry Fraichard and Christian Laugier. Path-velocity decomposition revisited and applied to dynamic trajectory planning. In *[1993] Proceedings IEEE International Conference on Robotics and Automation*, pages 40–45. IEEE, 1993.

[14] FSG. Fsg once again leads the world of formula student with a new competition class, formula student driverless (fsd). `https://www.formulastudent.de/pr/news/details/article/autonomous-driving-at-formula-student-germany-2017/`, 2016.

[15] Santiago Garrido, Luis Moreno, and Pedro U Lima. Robot formation motion planning using fast marching. *Robotics and Autonomous Systems*, 59(9):675–683, 2011.

[16] Alessandro Gasparetto, Paolo Boscariol, Albano Lanzutti, and Renato Vidoni. Path planning and trajectory planning algorithms: A general overview. In *Motion and operation planning of robotic systems*, pages 3–27. Springer, 2015.

[17] G. Genta. *Motor Vehicle Dynamics: Modeling and Simulation.* Advances in Fuzzy Systems. World Scientific, 1997.

[18] Gabriel M Hoffmann, Claire J Tomlin, Michael Montemerlo, and Sebastian Thrun. Autonomous automobile trajectory tracking for off-road driving: Controller design, experimental validation and racing. In *2007 American Control Conference*, pages 2296–2301. IEEE, 2007.

[19] SAE international. Taxonomy and definitions for terms related to driving automation systems for on-road motor vehicles. *SAE International,(J3016)*, 2016.

[20] Istituto Nazionale di Statistica Istat. Road accidents, year 2019. pages 1–3, 2020.

[21] Kazimierz Jamroz, Marcin Budzyński, Aleksandra Romanowska, Joanna Żukowska, Jacek Oskarbski, and Wojciech Kustra. Experiences and challenges in fatality reduction on polish roads. *Sustainability*, 11(4):959, 2019.

[22] Xj Jing. *Motion Planning*. BoD–Books on Demand, 2008.

[23] Juraj Kabzan, Miguel I Valls, Victor JF Reijgwart, Hubertus FC Hendrikx, Claas Ehmke, Manish Prajapat, Andreas Bühler, Nikhil Gosala, Mehak Gupta, Ramya Sivanesan, et al. Amz driverless: The full autonomous racing system. *Journal of Field Robotics*, 37(7):1267–1294, 2020.

[24] Oussama Khatib. The potential field approach and operational space formulation in robot control. In *Adaptive and Learning Systems*, pages 367–377. Springer, 1986.

[25] Pradeep Khosla and Richard Volpe. Superquadric artificial potentials for obstacle avoidance and approach. In *Proceedings. 1988 IEEE International Conference on Robotics and Automation*, pages 1778–1784. IEEE, 1988.

[26] John Ryan Kidd. Performance evaluation of the velodyne vlp-16 system for surface feature surveying. 2017.

[27] Daniel Koditschek. Exact robot navigation by means of potential functions: Some topological considerations. In *Proceedings. 1987 IEEE International Conference on Robotics and Automation*, volume 4, pages 1–6. IEEE, 1987.

[28] Jean-Claude Latombe. Motion planning: An overview. *Course notes*, 28:319, 1991.

[29] Steven M LaValle. Rapidly-exploring random trees: A new tool for path planning. 1998.

[30] Steven M LaValle. *Planning algorithms*. Cambridge university press, 2006.

[31] Todd Litman. Autonomous vehicle implementation predictions: Implications for transport planning. 2020.

[32] Tomás Lozano-Pérez and Michael A Wesley. An algorithm for planning collision-free paths among polyhedral obstacles. *Communications of the ACM*, 22(10):560–570, 1979.

[33] MATLAB™. fitpolynomialransac. fit polynomial to points using ransac. https://www.mathworks.com/help/vision/ref/fitpolynomialransac.html, 2016.

[34] MATLAB™. Vehicle body 3dof. 3dof rigid vehicle body to calculate longitudinal, lateral, and yaw motion. https://www.mathworks.com/help/vdynblks/ref/vehiclebody3dof.html, 2017.

[35] MATLAB™. Longitudinal controller stanley. control longitudinal velocity of vehicle by using stanley method. https://www.mathworks.com/help/driving/ref/longitudinalcontrollerstanley.html, 2018.

[36] World Health Organization. *Global status report on road safety 2015*. World Health Organization, 2015.

[37] Hans Sagan. Hilbert's space-filling curve. In *Space-filling curves*, pages 9–30. Springer, 1994.

[38] STEREOLABS. Zed stereo camera. https://www.stereolabs.com/zed/, 2020.

[39] FSG Formula Student. Official formula student rulebook, 2020.

[40] Osamu Takahashi and Robert J Schilling. Motion planning in a plane using generalized voronoi diagrams. *IEEE Transactions on robotics and automation*, 5(2):143–150, 1989.

[41] Charles W Warren. Global path planning using artificial potential fields. In *1989 IEEE International Conference on Robotics and Automation*, pages 316–317. IEEE Computer Society, 1989.

[42] Dino Živojević and Jasmin Velagić. Path planning for mobile robot using dubins-curve based rrt algorithm with differential constraints. In *2019 International Symposium ELMAR*, pages 139–142. IEEE, 2019.